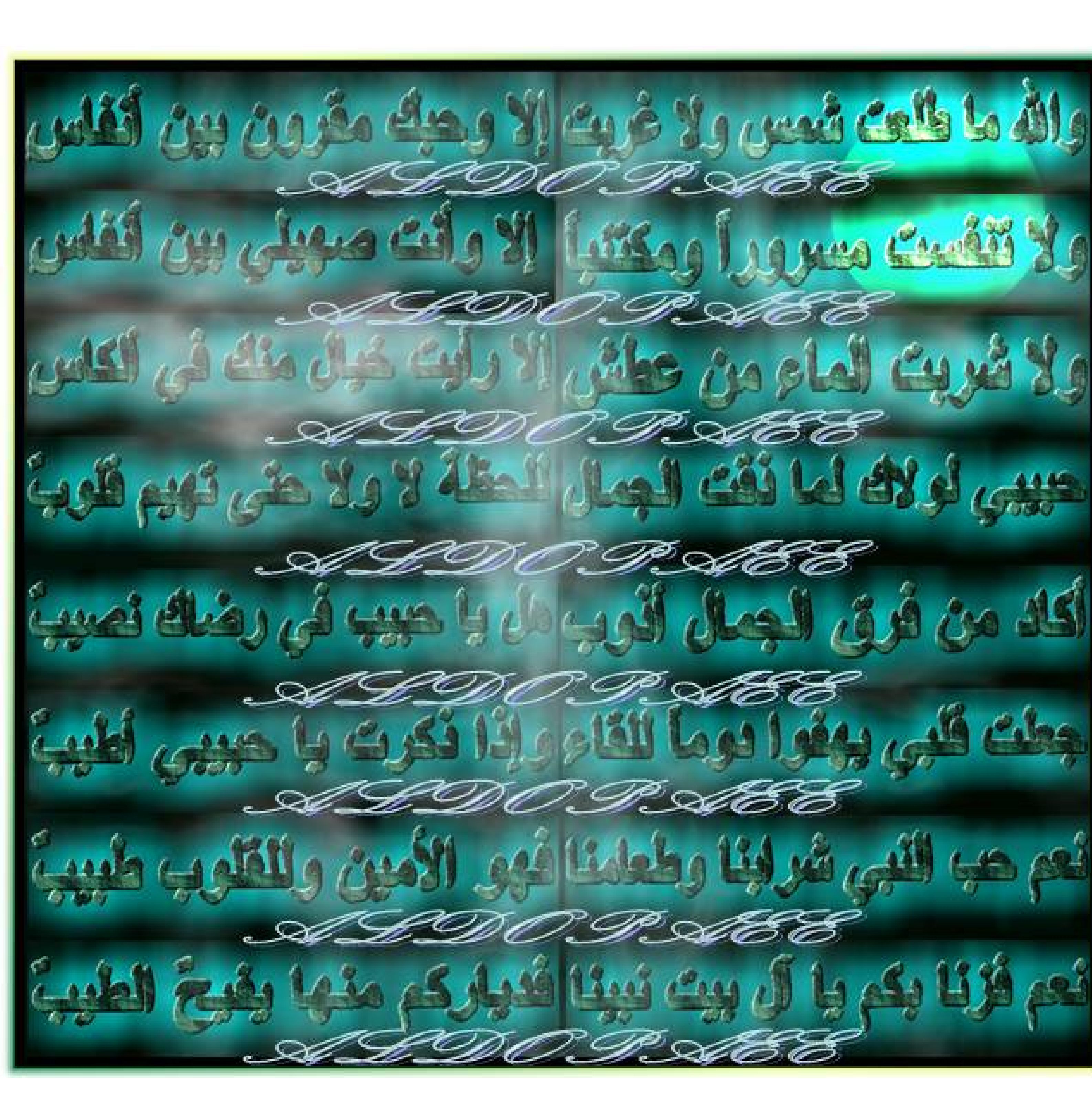

OOP IN JAVA
البرمجة الموجهة بلغة
OOP IN JAVA
اعداد/ محمد عيسى الدبعي
JAVA
عام الاصدار/ ٢٠٠٨م


والله ما طلعت شمس ولا غربت إلا وحبك مقرون بين أنفاس
ولا تنفست مسروراً ومكتئباً الا وأنت صهيلي بين أنفاس
ولا شربت الماء من عطش إلا رأيت خيال منك في الكاس
حبيبي لولاك لما نفت الجمال للحظة لا ولا حتى تهيم قلوبٌ
أكاد من فرق الجمال أذوب هل يا حبيب في رضاك نصيبٌ
جعلت قلبي يهفوا دوما للقاء وإذا ذكرت يا حبيبي أطيبٌ
نعم حب النبي شرابنا وطعامنا فهو الأمين وللقلوب طبيبٌ
نعم فزنا بكم يا آل بيت نبينا فدياركم منها يفوحَ الطيبٌ

رسالة شكر وعرفان

أهديها بباقة فل وريحان إلى الحبيب الغالي
الذي وقف القلم حائراً أمامه محاولاً ترتيب
الحروف ليكون منها كلمات تصف شرارة
من لهيب و بركان حبي لهُ....

د / أبو الغيث محجوب

إلي أخواني الذين لم تلدهم أمي وهم في أحشاء قلبي

- ☒ كرم باسودان
- ☒ سند علي عمر


الإهداء
إلى من أعطتني الرعاية دائماً
إلى من كانت بجواري دائماً
إلى ألبذره التي طالما روتني لأصبح ثمرة
أمي الحبيبة
إلى كل من يعشق نبينا محمد(صلى الله عليه وعلى اله)
إلى كل من علمني وتعب من اجلي
إلى ارض اليمن الحبيبة
تقبلوا هديتي

تأليف


A.DOPAEE


السيرة الذاتية
aldopaee@hotmail.com...a717@maktoob.com
الاسم: عمار محمد عيسى الدبعي
الجنسية: يمني
محل الإقامة: الجمهورية اليمنية
المؤهلات / طالب جامعي كلية علوم وهندسة حاسوب
www.a717jeeran.com

الفهرس

لمن هذا الكتاب

هذا الكتاب لمن قد تعلم لغة السي ++ أو أساسيات الجافا .

فكلا اللغتين متشابهتين بعض الشيء فتقيم الكتاب

- ✓ **متوسط**
- ✓ **متقدم**



المقدمة

الحمد لله رب العالمين حمداً يوافي نعمة ويكافئ مزبدة الحمد لله الذي خلق الظلمات والنور وصلى الله على سيد الخلق معلم الناس الخير حبيبنا وشفيعنا محمد بن عبد الله النبي الأمي الذي بذا برسالته ب(اقرأ باسم ربك الذي خلق الإنسان من علق إقراء وربك الأكرم الذي علم بالقلم) وختمها ب(اليوم أكملت لكم دينكم وأتممت عليكم نعمتي ورضيت لكم الإسلام دينا) صدق الله العظيم وبلغ رسوله الكريم ونحن على ذلك شاهدين من اليوم إلى يوم الدين أما بعد أخواني وأحباب قلبي يا من حييتهم من غير أن تراهم أعيني ابدآ بالسلام والتحية المباركة آلا وهي السلام عليكم ورحمة الله وبركاته .

الحمد لله الذي من علي بهذا الفضل العظيم أن علمني بالقلم وجعلني مسلما فمنحني الإرادة لانجاز هذا الكتاب فلقد أنشأته بسبب فقدان كتب هذه المادة على الانترنت وبالغة العربية واستحالة وجودها وأنا من عانيت من هذه المشكلة والسبب الأخر تيسيراُ لإخواني الطلاب الجامعيين الذين صعبت عليهم فهم هذه المادة وتبسيط لهم معنى هذه المادة وأنا متأكد أن كل من سيحصل على هذا الكتاب أنة سيفرح ويشكرني كثيراُ واعلموا أن كل ما كتبته هو خلاصة ولب البرمجة الموجهة و اتحدي أي مرجع يبين ما بينته لكم من دون بخل عليكم فكل ما بدهني أعطيته لكم ، املآ أن ينتفع به كل من يقراه أو كانت له حاجة في علوم الحاسوب . فيتناول هذا الكتاب موضوعات متعددة لوصف البرمجة الموجهة بلغة **oop in Java** ولقد وثقت هذه المواضيع ببرامج علمية طبقت جميعها للتأكد من صحتها وأيضاً وتقت بالرسوم البيانية لترسيخ الفكر في ذهن القارئ.

وأخيراُ نسأل الله أن يحقق هذا الكتاب الهدف الذي كتب لأجله ويعلم الله أن غايتي في هذا أن يعم الفائدة في ارض المسلمين وكل مسلم ومسلمة طالباُ منكم دعوه صالحة في ظهر الغيب وان تصلوا وتسلموا على من علمنا وأنبانا نبينا محمد حبيب قلبنا ألف ألف مليون صلاة وسلام من رب العباد عدد تحرير السطور وعدد المخلوقات والمخلوق صلاه دائمة من اليوم إلى يوم الدين .

واسأل الله أن يبارك لنا ولكم في كل ما كتبته و تعلمته وتعلمتموه .

والحمد لله .....

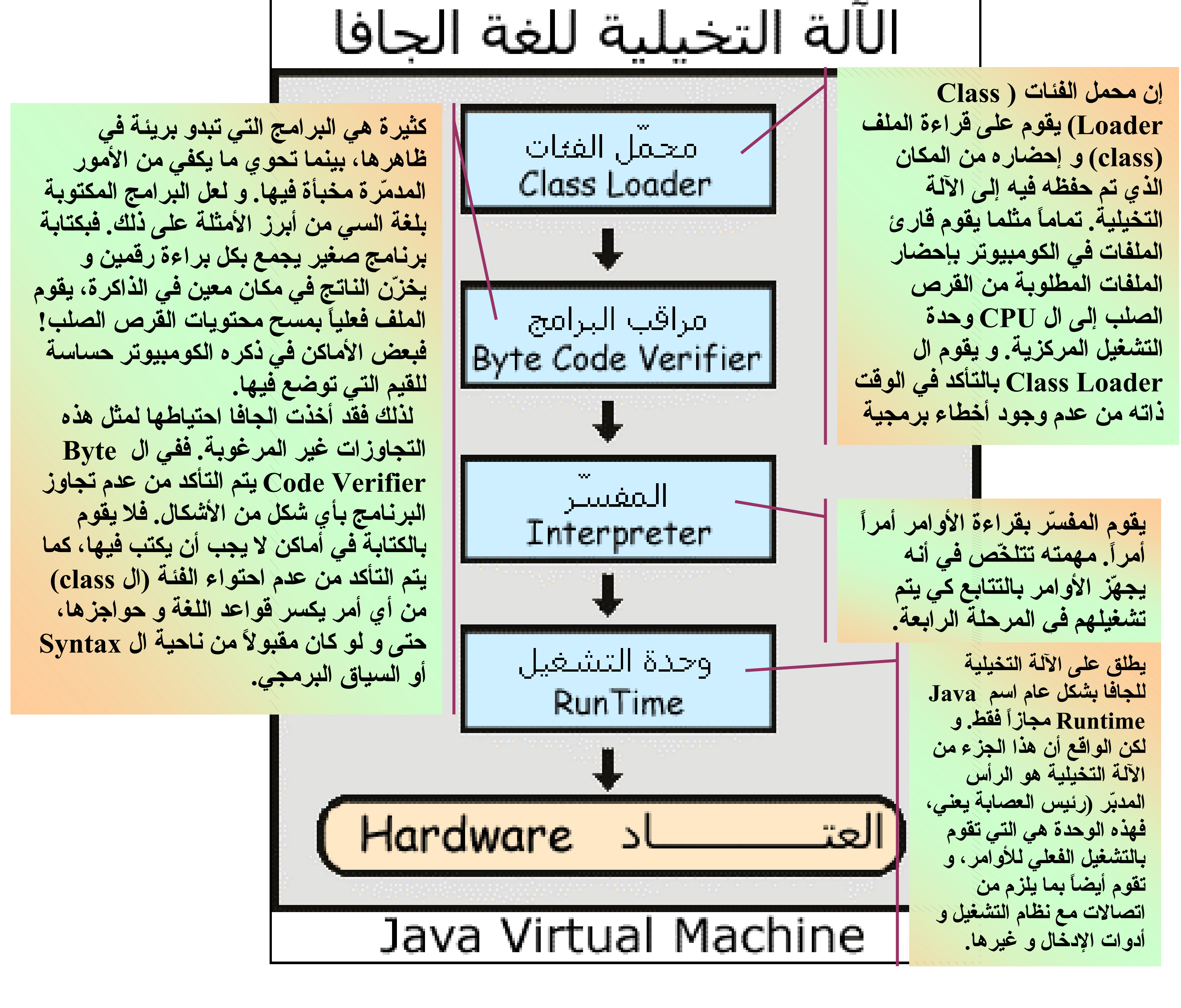
مقدمة عن لغة الجافا

كثيراً ما نسمع في هذه الأيام عن لغة الجافا حتى يبدو وكأنها قي كل مكان حتى في المكتبات لو تمعنت النظر لوجدت كتب المكتبة مليئة بكتب الجافا حتى الصحف الضخمة و المجلات تجد فيها العديد من المقالات التي تتحدث عن الجافا .
كل هذا يجعلك تتساءل عن سبب انتشار هذه اللغة والجواب ببساطة أنها تتيح للمستخدمين إمكانية تطوير تطبيقات تعمل على الويب من اجل المخدمات والأجهزة الصغيرة كالهاتف النقال وغيرة . وألان لنبحر سوياً قي تاريخ الجافا .
ففي أوائل التسعينيات من القرن العشرين ١٩٩١ اخترعت لغة الجافا شركة صن ميكروسيستمز ولهذا الاختراع قصة عجيبة حيث أن الشركة كانت قبل ذلك قد كلفت المهندس جيمس جوزلينج بوضع برامج لتشغيل الأجهزة التطبيقية الذكية مثل التليفزيون التفاعلي باستخدام لغة سي++ وحينها وجد جيمس جوزلينج صعوبة في التعامل مع هذه اللغة فقام هو وفريق العمل المساعد له بتطوير هذه اللغة فولدت لغة جديدة تتوافق مع احتياجاته فكانت لغة الجافا وقد خططت شركة صن في تلك الأيام لاستغلال هذه اللغة الوليدة في التليفزيون التفاعلي لكي تربح المليارات وحدث نوع من البطء في مشروع التلفزيون التفاعلي ربما عن قصد من الشركات الأخرى المنافسة ـ ونتيجة لذلك فكرت شركة صن في توقيف مشروع تطوير هذه اللغة الوليدة وتسريح العاملين في هذا المشروع أو نقلهم إلى قسم آخر ولكن حدث ما لم يكن في الحسبان حيث أنه في هذه الفترة كانت الانترنت قد بدأت في الانتشار بسرعة مذهلة مع نزول نظام الويندوز للأسواق وحيث أن لغة الجافا الوليدة التي اخترعت أصلا لبرمجة الأجهزة التطبيقية فيها من السمات ما يجعلها أكثر توافقا مع الشبكة العنكبوتية الدولية ـ الإنترنت ـ فقد كان لها السبق وأضافت الكثير إلى الإنترنت الذي كان قبلها مقصورا على تبادل النصوص ولكن المطورين بشركة صن ابتكروا طريقة تجعل برامج الجافا تعمل بسهولة في صفحات الإنترنت وغيروا الاسم الذي كان قد أطلقه عليه مبتكرها من آواك ـ شجرة السنديان ـ إلى الجافا ومن هنا أصبحت الجافا مرتبطة في شهرتها بالإنترنت حيث أن برنامج جافا صغير يوضع في صفحة من صفحات موقع على الشبكة الدولية يراه الملايين في جميع أنحاء العالم في نفس الوقت وقد كان هذا لا يتوفر إلا مع الجافا مما أعطاها شهرة واسعة أكبر من شهرة نجوم هوليود ولحسن حظ شركة صن أن لغة الجافا أكدت نفسها في المجال الذي طورت له أصلا فقد بدأ الآن التليفزيون التفاعلي في الانتشار وما يسمى سينما المنزل والمشاهدة حسب الطلب وليس هذا فقط بل أنتشر ما هو أكثر فائدة لشركة صن وهو الهاتف المحمول وللجافا أكبر دور في برمجة البرامج التي يعمل بها في أجياله السابقة واللاحقة ولا نستغرب أن يحدث نوع من الغيرة بين شركة ميكروسوفت وشركة صن ميكروسيستمز مما دفع ميكروسوفت إلى أن تحذف ماكينة الجافا الافتراضية من الإصدارة الأولى للويندوز اكس بي وهذه الماكينة الافتراضية مسئولة عن عرض برامج الجافا على الانترنت ولكن ميكروسوفت تراجعت أمام طلب ملايين المستخدمين حول العالم فوضعتها مرة ثانية في الإصدارات اللاحقة وقد كانت قضية مشهورة تناولتها الصحف والمجلات .
وقد ساهم في شهرة الجافا أيضا برامجها العلمية التفاعلية التي تصلح لمعظم المناهج التعليمية في جميع مراحل التعليم وبالتالي فان لها دورا كبيرا في التعليم الالكتروني والتعليم عن بعد والفصول الافتراضية .

مميزات الجافا

١. لغة سهلة التعلم و كبيرة الأمكانيات وبدون تعقيدات كالسي بلس بلس .
٢. لغة برمجية تعمل بواسطة الأهداف OOP فهي الرائدة في تقنية الـ OOP أو برمجة المتجهات و تعتبر أكثر لغة تطبق الفكرة كأحد مميزاتها الجبارة .
٣. لغة أمنة من ناحية البرامج التي تنفد في الحاسب فإنها لا تؤدي نظام التشغيل .
٤. لها بيئة تشغيل خاصة بها . JVM .
٥. لها مكتبة فصائل . Class Libraries .
٦. تقوم على لغةC ++C .
٧. يمكن لأي برنامج معمول بلغة الجافا أن يعمل بشكل مباشر على أي framework بمعنى إن البرنامج يمكن أن يعمل على Windows Xp أو Linux أو Mac على عكس إمكانيات لغات البرمجة الأخرى مثل C++ آو حتىC# .
٨. تأخذ تلقائياً البيئة التي تعمل ضمنها وتدعى هذه التقنية تقنية سوينغ أي انك عندما تقوم بتطوير جافا فان هذه التطبيقات عندما تعمل ضمن ويندوز فان عناصرها المختلفة تأخذ شكل ويندوز وعندما تعمل ضمن بيئة الماكنتوش فأنها تأخذ تلقائياً شكل واجهات الماكنتوش وهذه ناحية هامه جداً للمستخدم وللمبرمج
٩. والكثير من المميزات التي لم أتطرق إليها

ولكن ماهو سر هذه القوة....
لقد كان في فكر مخترع هذه اللغة هو اختراع لغة تستطيع أن تركز بها في وصف المشكلة التي تريد حلها بعيدا عن تفاصيل نظام التشغيل. هذه التفاصيل مثل:
كيفية كتابة ملف على القرص الصلب
كيف أكتب معلومات في ذاكرة الجهاز وكيف أعيد قراءتها
كيف أخلق المعلومات في صورة Object ومتى أقوم بحذفها
فمثلا إذا كنت أريد أن أكتب برنامج لإدارة المدارس وقلت لك أن تأخذ التفاصيل السابقة معك وأن تفكر في المشكلة فلن تصل لحل المشكلة مثل شخص آخر يضع كل تفكيره وتركيزه في وصف نظام إدارة المدرسة مثل من له حق استخدام النظام و ما هي المعلومات المطلوبة عن المدرسين و الطلبة و المناهج و ما هي السيناريوهات USE CASES المختلفة للنظام.
إذن فلغة الجافا هي تقريبا مثل أي لغة طبيعية كالعربية والإنجليزية نستخدمها لنعبر عن أفكارنا ومشاكلنا ونتواصل بها مع الآخرين .



تعتبر JVM الجزء الوحيد من بيئة البرمجة الجافية الذي يعرف ما هو نظام التشغيل الذي تعمل عليه البرامج المختلفة. فالفئات كما ذكرت، تعرف أنها يجب أن تعمل لحساب الآلة التخيلية. و هي واحدة في كل مكان مهما اختلف نظام التشغيل و نوع الكومبيوتر. أما الآلة التخيلية نفسها، فهي العضو الذي يتصل بالكومبيوتر.. و يقوم بما يلزم من عرض على الشاشة، أو قراءة من الكيبورد، إذا لابد أن تعرف ال JVM عن نظام التشغيل الذي تعمل عليه.

الفرق بين الجافا والسى ++

Java	C++
تستخدم الأهداف بشكل أساسي في كتابة البرامج ولا يوجد برنامج خالي من الأهداف class	لا تستخدم الأهداف في معظم برامجها
أبعدت شيء أسمة المؤشرات بسبب تعقيدها وكثرة أخطائها إن لم تعامل صحيحاً .	تدعم المؤشرات
لا تدعم الوراثة المتعددة وقد أبدلتها بشيء اسمه الواجهات interface	تدعم الوراثة المتعددة
لا يوجد مدمرات Destructor function في الوراثة	يوجد في الوراثة
تعمل على أي Operating System	نظير السابق
البرامج المكتوبة بالـ java و مع الأسف محكوم عليها بالبطئ الشديد كحال أشهر البرامج في العالم برامج الإدارة في (Oracle و (Builder) و (JDeveloper) في أحد المرات سئل James Gosling مخترع الجافا عن سبب بطي الجافا فأجاب : إن الديكتاتورية أسرع دائما من الديمقراطية فعلا إجابة نموذجية ، السرعة ليست كل شيء .	سريعة جدا بمعدل ١٠-٢٠ مرة من الجافا و البرامج المكتوبة بها تكون سريعة جدا كذلك
لا تدعم البرمجة الجينية أي template	اللغة الوحيدة التي دعمت هذه الخاصية
تدعم التجريد abstraction	لا تدعم التجريد abstraction
لا يوجد شيء أسمة تجاهل العمليات operator	تدعم هذه الخاصية
تدعم آلية التزامن synchronized	لا تدعم إلية التزامن
لا تسمح بتمرير المعطيات بواسطة المرجع reference	تدعم هذه الخاصية
لا تستطيع التعامل مع مواقع ذاكريه أو مع المنافذ فلا تستطيع كتابة كود ليشغل قطع .	تعتبر من اللغات المشهورة بالتعامل مع الهاردوير
تملك دعم خيالي للويب فهي لغة الويب والشبكات	تدعم بشكل بسيط عن نظيرها
يجب حفظ البرنامج باسم الكلاس الرئيسي	ليس مهماً
يتم إدارة العمليات في الذاكرة تلقائيا فتكون أكثر أمانا .	يتم يدوياً .

أفضل مترجم للجافا

توجد أكثر من طريقة لكتابة برامج الجافا وترجمتها منها :
(1)استعمال المكتبة JDK مباشرة مع استعمال أي محرر سطور:
تعتبر هذه الطريقة التقليدية هي استعمال أدوات JDK التي أنتجتها شركة SUN مع أي محرر سطور لإعداد البرنامج وهي الطريقة المتبعة عند شرح أجزاء لغة جافا ونبدأ كما يلي :
الأدوات المطلوبة لاستعمال هذه الطريقة:
1- محرر سطور وليكن " Notepad المفكرة " الموجود مع ويندوز .
2 - مجموعة :JDK ويمكنك الحصول على مكتبة JDK من موقع. SUN من هذا الرابط
http://java.sun.com/j2se/downloads.html
3- أدوات المجموعة :JDK
- الملف : Javac وهو الملف التنفيذي المستعمل في ترجمة الملف المصدر إلى الصورة التنفيذية .
- الملف :Java هو البرنامج المسئول عن تنفيذ برامج **Java** التنفيذية بعد تحويلها .
- الملف **Applet** :Viewer لعرض برنامج **Applet** للإخبار.

(2)استعمال برامج وسيطة مثل JCreator وأنا أفضلة واستخدمه و يمكنك إنزالها من هنا
(3) http://www.jcreator.com/Download.htm
استعمال البرامج المعدّة للغة الجافا مثل : JBuilder Forte : يمكنك إنزالها من هنا
http://www.borland.com/jbuilder/personal

http://www.jinfonet.com/download/Forte

بعض الأساسيات فى الجافا

أنواع البيانات

النوع	الحجم/ Format	المواصفات (الصحيحة الأرقام)
byte	8-bit two's complement ٨ بت من مضاعفات العدد ٢	-127 to 127
short	16-bit two's	-32768 to 35767

	complement	نستخدمه للأعداد الصحيحة الأقل أو المساوية للعدد ٢ مرفوع للأس ١٦
int	32-bit two's complement	للإعداد الصحيحة الأقل أو المساوية للعدد ٢ مرفوع للأس ٣٢ بت
long	64-bit two's complement	**Long integer** للأعداد الصحيحة الأقل أو المساوية للعدد ٢ مرفوع للأس ٦٤ بت
		الأرقام العشرية
float	32-bit IEEE 754	**Single-precision floating point** الأرقام العشرية الأقل أو المساوية للعدد ٢ مرفوع للأس ٣٢ بت وعند تعريف متغير من هذا النوع يجب وضع حرف f في نهاية الرقم وإلا اعتبرتها لغة الجافا من نوع double
double	64-bit IEEE 754	**Double-precision floating point** الأرقام العشرية الأقل أو المساوية للعدد ٢ مرفوع للأس ٦٤ بت
		(أخرى أنواع)
char	16-bit Unicode character	**A single character** حرف واحد
Boolean	true or false	**A Boolean value (true or false)** قيمة "بولن" صحيح أو غير صحيح

*** المؤثرات operators**

المؤثرات هي الرموز التي تربط بين المتغيرات والثوابت لإنشاء علامة أو معادلة تختلف أنواع المؤثرات باختلاف وظيفة كل مؤثر .

١ـ المؤثرات الحسابية arithmetic operators

+ علامات الجمع addition

- علامات الطرح Subtraction

* علامات الضرب multiplication
/ علامات القسمة division
وتستخدم مع المتغيرات والثوابت الرقمية

٢- مؤثرات المقارنة Relational operators:-

وتستخدم لمقارنة قيمتين :

المؤثر	الرمز	مثال	النتيجة
أكبر من greater than	>	10 > 8	1
أصفر من less than	<	10 < 8	٠
يساوى equal to	==	١٠==٨	0
لا يساوى not equal to	! =	١٠ !=٨	1
أقل من أو يساوى less than or equal to	<=	١٠ < = 8	0
أكبر من أو يساوى greater than or equal to	>=	10 > = 8	٠

٣- المؤثرات المنطقية Logical operator

المؤثر	الرمز	مثال	النتيجة
و AND	&&	10 > 8 && 9 > 7	1
أو OR	||	1 0 < 8 || 7 < 8	١
لا NOT	!	! (10 = = 8)	1

٤- مؤثرات التخصيص Assignment Operators

وهى مؤثرات تخزين قيمة في متغير فمثلا إذا كانت قيمة ٦ = ٩

التخصيص التقليدي	الطريقة الحديثة	النتيجة	المؤثر
A = a +5	A + = 5	11	+ = addition assignment operators
A = a – 5	A - = 5	1	Subtraition assignment opertors
A = a + 5	A * = 5	30	Multipication assibnment operators
A = a / 3	A / = 3	2	Division assignment operators

٥- مؤثرات الزيادة والنقصان Decrement &increment

A = a + 1	A ++	7	مؤثر زيادة واحد

مؤثر نقصان واحد	5	A --	A = a – 1

٦- مؤثر باقي خارج القسمة %

يستخدم لمعرفة باقي القسمة (لتحديد هل الأرقام الموجودة في المتغير زوجية أو فردية فمثلا إذا كانت قيمة a = 5 وكتب C = a % 2 يكون باقي الرقم 2 / 5 = 1

التعبير Expression

التعابير هي أساس إي شفرة برمجية ، بالتعاون مع الاساسيات الأخرى للغة جافا نستخدم التعابير لحساب قيم المتغيرات وتحليل النتيجة وذلك حتى نستطيع التحكم في طريقة سير وعمل البرنامج. ويتم ذلك عن طريق حساب القيمة وإرجاعها للكمبيوتر للقيام بفعل معين.

باختصار التعابير هي عبارة عن مجموعة متغيرات ومشغلات وأوامر لحساب قيمة معينة.

شروط كتابة البرامج بالجافا

1. **يجب تسمية البرنامج بنفس اسم الكلاس المستخدم**
2. **الانتباه من كتابة بعض الكلمات المحجوزة بحرف صغير فيجب كتابة أول حرف لهذه الكلمة بحرف كبتل مثل System.out.print(); وهي أمر الطباعة .**

وهذا ابسط كود يطبع العبارة التي بداخل علامة التنصيص

```
public class y{
   public static void main(String[] args) {
     System.out.println("Aldopaee");
   }
}
```

المتغيرات

هي أنواع من المعلومات التي يمكن تخزينها في موقع خاص في البرنامج ونستطيع تغييرها أثناء عمل البرنامج
المتغيرات هي الوسيلة التي يتذكر بها الحاسب القيم خلال تشغيل البرامج.

أنواع المتغيرات

- **متغيرات من نوع static / هذه المتغيرات نستطيع أن نقول أنها ن نوع ثابت أي ليس بمعنى أننا لا نستطيع تغير قيمتها ولكن قد تكون بداخل الكلاس فنتعامل نعها مباشرة أي كأنها عامة فعند ترجمة البرنامج يتعرف عليه المترجم مباشرة قبل الدخول إلى الدالة الرئيسية .**
- **متغيرات محلية local / وهي المتغيرات المعرفة على مستوى البلوك أي المقطع ولا نستطيع الوصول إليها من خارج البلوك أي {} فعندما نخرج من البلوك فإنها تدمر من الذاكرة وتنتهي حياتها .**
- **معاملات / وهي التي تم تعريفها في رأس الدالة فتبدءا حياتها عند تنشيط الدالة وتنتهي حياتها عند انتهاء الدالة .**

مجال تغطية المتغيرات
وهو الجزء من البرنامج الذي نستطيع من خلاله الوصول إلى المتغير .فتسمى المتغيرات المعرفة داخل المنهج بالمتغيرات المحلية .

- ❖ **يجب الإعلان عن المتغيرات قبل استخدامها .**
- ❖ **المتغير المعلن برأس الحلقة for يمكن استخدامه داخل الحلقة فقط بخلاف لغة السي حيث بعد تعريفه و يمكن استخدامه خارج الحلقة فهذا الكود من الأخطاء الشائعة**

```
for(int i=0;i<4;i++){}
System.out.println(i);
```

- ❖ **هذا الكود من الأخطاء الشائعة والسبب أن I لم يتم التعرف علية فتصحيح هذا الخطاء نجعل I من نوع ستاتك** **static int I=507;**

```
public class y{
   int i=507;
   public static void main(String[] args) {
   int b = i;
   System.out.println(b);
   }
}
```

- **ما خرج هذا الكود**

```
class ammar{
```

```
        static int a(){return j;}
        static int i=a();
        static int j=1;
        }
class aldopaee{
public static void main(String args[]) {
System.out.println(ammar.i);
}
}
```

التحويلات في الأنماط العددية

بمعنى التبديل بين أنواع البيانات مثل القناع وأحيانا يُجبر المبرمج في استخدام هذه العملية.

- **قاعدة**

- **عند تحويل نمط صغير إلى كبير فان المترجم تلقائياً يقوم بهذه العملية .**

```
Byte i=10;
Long k=20;
K=I;
```

- **عند تحويل نمط كبير إلى صغير فأنة من الضروري عمل casting أي قناع مثل**

```
Int i=256;
Byte b=(byte)I;
```

هنا يتم أخذ ٨ بت فقط ويتم إسنادها إلى b فتكون قيمة b=0 .

كيف تمت العملية ؟

إليكم هذه الطريقة التي ابتكرتها وسهلتها لكم ولم تذكر في أي مرجع

```
{
 {
  If (i<=127) b=I;
  Else
  If (i>127&bit [8].I=0) b= (sum bit [1] ->bit [7]). I
  Else
  If (i>127&bit [8].I=1) b= ((sum bit [1] ->bit [7]). I) -128
 }
 If (I<0) b*=-1;
}
```

إذا كان i اقل من 127 أي يملئ بايت واحد فان b يأخذ القيمة مباشرتاً

٠	١	١	١	١	١	١	١

أم إذا كان اكبر من ١٢٧ أي يفيض عن البايت الأول

٠																							١	٠	١	١	١	١	٠	١	١

و البت ٨ = ٠ أي غير مؤشر
فإننا نقوم بجمع محتوى البت الأول إلى السابع أي تحويل من ثنائي إلى عشري فنحصل على العدد ١٢٣ فيكون b=123

أم إذا كان اكبر من ١٢٧ أي يفيض عن البايت الأول و البت ٨ = ١ أي مؤشر

٠																							١	١	١	١	١	١	٠	١	١

محتوى I بالعشري يساوي 507 وعند تحول البايت الأول إلى السابع من الثنائي إلى العشري فإننا نحصل على العدد ١٢٣ وبطرحة من ١٢٨ فيكون b=-5
واليكم هذا مثال نفذوه وتأكدوا من صحة قولي

```
public class y{
   public static void main(String[] args) {
   int i=507;
   byte b=(byte)i;
  System.out.println(b);
   }


}
```



جميع الشروط التي ذكرتها في حالة القيم الموجبة أم إذا كانت قيمة I=-507 فإننا نطبق الخطوات السابقة تم نضرب b*=-1

١																							١	١	١	١	١	١	٠	١	١

* عندما يكون لديك عدد مؤشر أي سالب وتريد تمثيله بالثنائي فما عليك إلى أن تعتبر العدد موجب و تحول العدد إلى الثنائي تم تجد المتمم الأحادي له تم تضيف له واحد ويصبح عدد مؤشر بالثنائي ، وخلاصة القول يمكن أن تحول العدد إلى المتمم الثنائي مباشرة .
محتوى I بالعشري يساوي 507- وعند تحول البايت الأول إلى السابع من الثنائي إلى العشري فإننا نحصل على العدد ١٢٣ وبطرحة من ١٢٨ فيكون b = -5 وبضرب b*=-1 فتكون قيمة B=5 .

- في حالة تحويل نمط من float إلى int فإننا نبعد الكسور فقط

```
float i=256.6f;
int b=(int)i;
```

فتكون قيمة b=256.

- في هذا الكود سيقوم المترجم بإصدار خطاء

```
int b=200;
short i=b;
```

والسبب إن b مكون من ٤ بايت و I من ٢ بايت وهذه العملية تحتاج إلى تحويل .

- وهذا كود شامل لما سبق

```
class Conversion {
public static void main(String args[]) {
byte b;
int i = 257;
double d = 323.142;
System.out.println("\nConversion
of int to byte.");
b = (byte) i;
System.out.println("i and b " + i + " " + b);
System.out.println("\nConversion of double to int.");
i = (int) d;
System.out.println("d and i " + d + " " + i);
System.out.println("\nConversion of double to byte.");
b = (byte) d;
System.out.println("d and b " + d + " " + b);
}
}
```

```
D:\JCreator LE\GE2001.exe

Conversion of int to byte.
i and b 257 1

Conversion of double to int.
d and i 323.142 323

Conversion of double to byte.
d and b 323.142 67
Press any key to continue...
```

```
class y {
public static void main(String args[]){
byte b = 42؛
char c = 'a؛'
short s = 1024؛
int i = 50000؛
float f = 5.67f؛
double d = .1234؛
double result = (f * b) + (i / c) - (d * s)؛
System.out.println((f * b) + " + " + (i / c) + " - " + (d * s));
System.out.println("result = " + result)؛
}
}
```

```
D:\JCreator LE\GE2001.exe
238.14 + 515 - 126.3616
result = 626.7784146484375
Press any key to continue..._
```

- **قاعدة / أي عملية على byte يجب عمل تحويل وإلا ينتج خطاء فهذا الكود خاطئ**
فتصحيح هذا الخطاء نعمل Casting b=(byte) (b+5)

```
class aldopaee {
public static void main(String args[]){
byte b = 42;   b=b+5;
System.out.println(b);
}
}
```

- **هذا الكود خاطئ لماذا ؟**

```
class aldopaee {
public static void main(String args[]){
    byte a;
    System.out.println(a);
}
}
```

والسبب أن لغة الجافا لا تسند قيم افتراضية للمتغيرات بخلا ف لغة السي .
وهنا الخطأ أنة لم يتم تهيئة a بقيمة .

الاقترانات الخاصة بالسلاسل

إيجاد طول السلسلة الرمزية:	
`s.length()`	ترجع الطريقة **length()** طول السلسلة الرمزية **s**.
عمليات المقارنة بين سلستين رمزيتين (ملاحظة: لاتستخدم **==** و **!=**).	
`s.compareTo(t)`	تقوم الطريقة بمقارنة السلسلة الرمزية **s** مع السلسلة الرمزية **t** وتعيد رقم سالب اذا كانت **s** اقل من **t** وتعيد صفر إذا كانت **s** تساوي **t** وتعيد رقم موجب إذا كانت **s** أكبر من **t**.
`s.compareToIgnoreCase(t)`	تعمل هذه الطريقة بنفس عمل الطريقة **compareTo()** ولكن مع اهمال حالة الحروف (صغيرة أم كبيرة).
`s.equals(t)`	تعيد **true** إذا كان **s** يساوي **t**.
`s.equalsIgnoreCase(t)`	تعمل هذه الطريقة بنفس عمل الطريقة **equals()** ولكن مع إهمال حالة الحروف (صغيرة أم كبيرة).
`s.startsWith(t)`	تعيد **true** إذا كان **s** يبدأ بالسلسلة الرمزية **t**.
`s.startsWith(t, i)`	تعيد **true** إذا كانت السلسلة الرمزية **t** موجودة في **s** بدءاً من الموقع **i**.
`s.endsWith(t)`	تعيد **true** إذا كان **s** تنتهي بـ **t**.

عمليات البحث: كل طرق () indexOf تقوم بإرجاع 1- إذا كان العنصر المراد البحث عنه غير موجود، ويمكن للعنصر المراد البحث عنه أن يكون حرف أو سلسلة رمزية.	
`s.indexOf(t)`	ترجع موقع أول مكان توجد فيه **t** داخل السلسلة الرمزية **s**.
`s.indexOf(t, i)`	ترجع موقع أول مكان توجد فيه **t** داخل السلسلة الرمزية **s** بعد الموقع **i**.
`s.indexOf(c)`	ترجع موقع أول مكان يوجد فيه الحرف المخزن في المتغير **c** داخل السلسلة الرمزية **s**.
`s.indexOf(c, i)`	ترجع موقع أول مكان يوجد فيه الحرف المخزن في المتغير **c** داخل السلسلة الرمزية **s** بعد الموقع **i**.
`s.lastIndexOf(c)`	ترجع موقع آخر مكان يوجد فيه الحرف المخزن في المتغير **c** داخل السلسلة الرمزية **s**.
`s.lastIndexOf(t)`	ترجع موقع آخر مكان توجد فيه السلسلة الرمزية **t** داخل السلسلة الرمزية **s**.

عمليات أخذ جزء من السلسلة الرمزية **string**.	
`s.charAt(i)`	ترجع الحرف الموجود في الموقع **i** داخل السلسلة الرمزية **s**.
`s.substring(i)`	ترجع جزء من السلسلة الرمزية **s** بدءاً من الموقع **i** وحتى النهاية.
`s.substring(i, j)`	ترجع جزء من السلسلة الرمزية **s** بدءاً من الموقع **i** وحتى الموقع **j-1**.
عمليات التعديل على السلسلة الرمزية **string** وإنشاء سلسلة رمزية جديدة.	
`s.toLowerCase()`	إنشاء سلسلة رمزية جديدة تحتوي كل ما في السلسلة الرمزية **s** بعد تحويل كل الحروف إلى حروف صغيرة.
`s.toUpperCase()`	إنشاء سلسلة رمزية جديدة تحتوي كل ما في السلسلة الرمزية **s** بعد تحويل كل الحروف إلى حروف كبيرة.
`s.trim()`	إنشاء سلسلة رمزية جديدة من السلسلة الرمزية **s** بعد الفارغ من البداية والنهاية.
`s.replace(c1, c2)`	إنشاء سلسلة رمزية جديدة من السلسلة الرمزية **s** بعد تبديل كل **c1** بـ **c2**، وهما من نوع **char**.
عمليات أخرى على السلاسل الرمزية **string**.	
`s.matches(regexStr)`	ترجع هذه الطريقة **true** إذا كانت السلسلة

	الرمزية **regexStr** تطابق السلسلة الرمزية **s** كاملة.
`s.replaceAll(regexStr, t)`	إنشاء سلسلة رمزية **string** جديدة بعد تبديل كل **regexStr** بـ **t**.
`s.replaceFirst(regexStr, t)`	إنشاء سلسلة رمزية **string** جديدة بعد تبديل أول **regexStr** بـ **t**.
`s.split(regexStr)`	إنشاء مصفوفة تحتوي على أجزاء من السلسلة الرمزية **s** مقسمة حسب ظهور **regexStr**.
`s.split(regexStr, count)`	كما في الطريقة **split(regexStr)** لكن مع تحديد عدد مرات التقسيم.

- **هذا الكود يبين الفرق بين مواقع المتغيرات و قيم المتغيرات في المساواة**

```
public class aldopaee {
  public static void main(String[] args) {
    Integer n1 = new Integer(47);
    Integer n2 = new Integer(47);
    System.out.println(n1 == n2);
    System.out.println((n1).equals(n2));
  }
}
```

هنا معناه هل موقع المتغير الأول نفس موقع المتغير الثاني

هنا معناه هل قيمة المتغير الأول تساوي قيمة المتغير الثاني بواسطة هذه الدالة

```
D:\JC
false
true
```

- **ما ناتج هذا الكود**

```
class Value { int i;}

public class aldopaee {
  public static void main(String[] args) {
    Value v1 = new Value();
    Value v2 = new Value();
    v1.i = v2.i = 100;
    System.out.println(v1.equals(v2));
  }
```

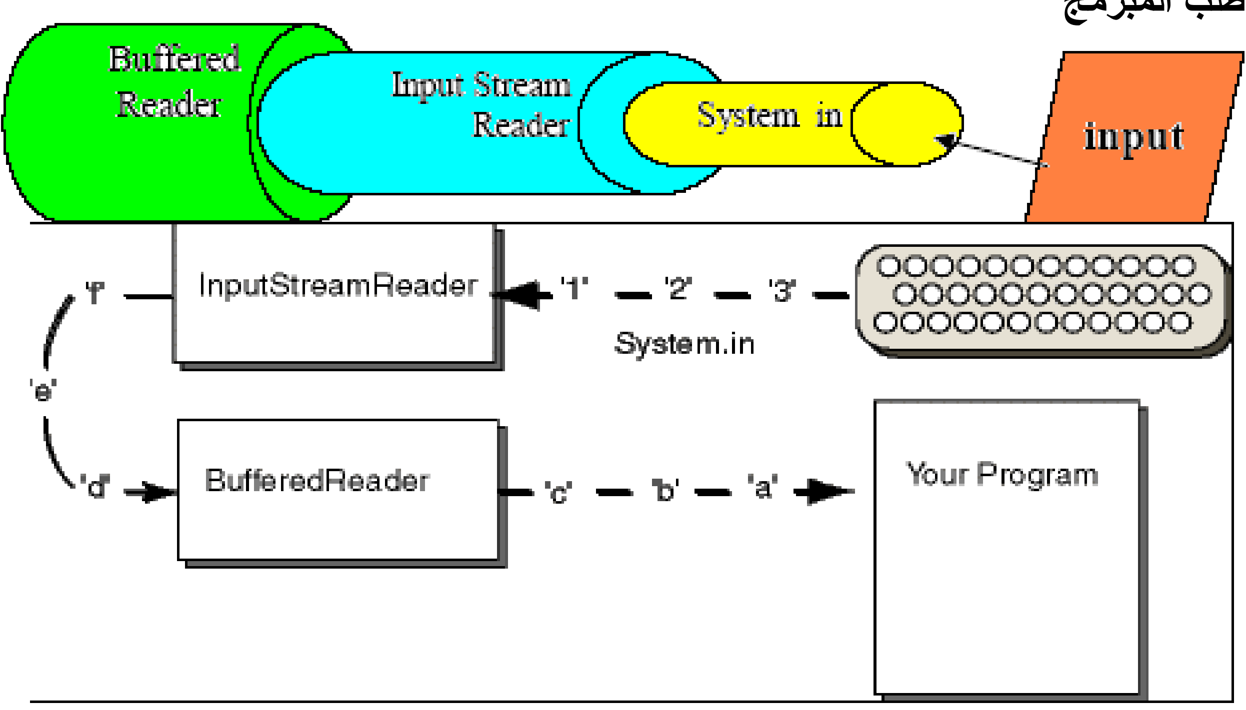
}

الدخل Input

غالباً أكثر الكتب تترك هذا الفصل إلى نهاية الكتاب ولكن نظراً لاحتياج القارئ إلى إدخال البيانات من لوحة المفاتيح بشكل مبكر فعجلت بشرحه لأنني من الذين عانوا من كيفه إدخال البيانات.

نلاحظ لغة الجافا تختلف عن بقية اللغات المشهورة كالسي بالادحال ففي السي نجد سهولة تامة باستخدام أوامر الإدخال دون تعقيد ولكن في الجافا تعد الإدخال في مراحل شبة معقدة .

تقدم لغة الجافا مجاري مؤقتة (buffered) التي تستخدم مصفوفة مؤلفة من البايتات أو من المحارف على حسب طلب المبرمج



ومن خلال الأشكال السابقة نلاحظ أن الإدخال بلغة الجافا يتكون من ثلاثة أنابيب

System.in

وهو الأنبوب الأول ويعمل على قراءة بايت واحد في كل مرة

Input StreamReader

ويعمل على تحويل كل ٢ بايت إلى حرف أو رمز

BufferedReader

ويعمل على تجميع هذه الحروف أو الرموز في الذاكرة المؤقتة لعمل منها سلسلة

```
BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
```

هنا القارئ br هو من نوع الكلاس BufferedReader والكلاس BufferedReader لة المعامل InputStreamReader والكلاس InputStreamReader له المعامل System.in وهو كلاس للقراءة بايت من لوحة المفاتيح .

١. وأول ما نقوم بة عند أي عملية الإدخال نستدعي مكتبة الإدخال في بداية البرنامج

```
Import java.io.*;
```

٢. نكتب إلية الإدخال التي نريدها وليكن إدخال عدة بايتات

```
BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
```

٣. في دالة main نكتب استثناء الإدخال

```
public static void main(String args[]) throws IOException
```

سيتم شرح الاستثناءات فيما بعد .

*** وننوه إلى أحبائي الطلاب أنة دائماً يتم قراءة البيانات من لوحة المفاتيح بصيغة أسكي إذا كانت القراءة بايت واحد وبصيغة سلسلة إذا كانت القراءة بعدة بايتات وعلى المبرمج تحويل من سلسلة نصية إلى أرقام عددية int باستخدام الدوال الخاصة بالسلاسل التي ذكرت سابقا .**

وهذا كود يبين كيفية الإدخال باستخدام بايت واحد

```
import java.io.*;
class y {
public static void main(String args[])throws IOException {
int b;
b=System.in.read();
System.out.println(b);
}
}
```

وهذا مثال أخر

```
import java.io.*;
class BRRead {
public static void main(String args[])throws IOException{
char c;
```

```
BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
System.out.println("Enter characters 'q' to quit.");
// read characters
do {
c = (char) br.read();
System.out.println(c);
} while(c != 'q');
}
}
```

هنا عملنا عملية تحويل نمط العدد من أسكي إلى محرف وقد بينا ذلك سابقاً أنة يتم القراءة بشكل أسكي إذا كان بايت واحد

وهذا كود لإدخال سلسلة مكونة من عدة بايتات

```
import java.io.*;
class y {
public static void main(String args[]) throws IOException
{String c;
BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
System.out.println("Enter characters 'q' to quit.");
c = br.readLine();
System.out.println(c);
}
}
```

نلاحظ الاختلاف هو عندما يقرءا بايت فإننا نعرف المتغير من نوع char وجملة القراءة read().

أم عندما نقري عدة بايتات فإننا نعرف المتغير من نوع String وجملة القراءة readLinr(). أي سطر كامل .

واليكم المفاجئة بهذا الكود لكيفية الإدخال بلغة الجافا

انسخوا الكود وجربوه وشاهدو النتيجة

```
import javax.swing.*;
public class aldopaeeinput{
public static void main(String args[]){
 String s;
 int b;
 s=JOptionPane.showInputDialog("Enter a number:");
 b=Integer.parseInt(s);
 System.out.println(b*b);
}
}
```

مكتبة الإدخال من نوع متطور فجوال

آلية الإدخال نفس السابق

دالة تقوم بتحويل سلسلة رقمية إلى عددية

- ها أليست لغة الجافا لغة جميلة !!!!!!!!!!!!!

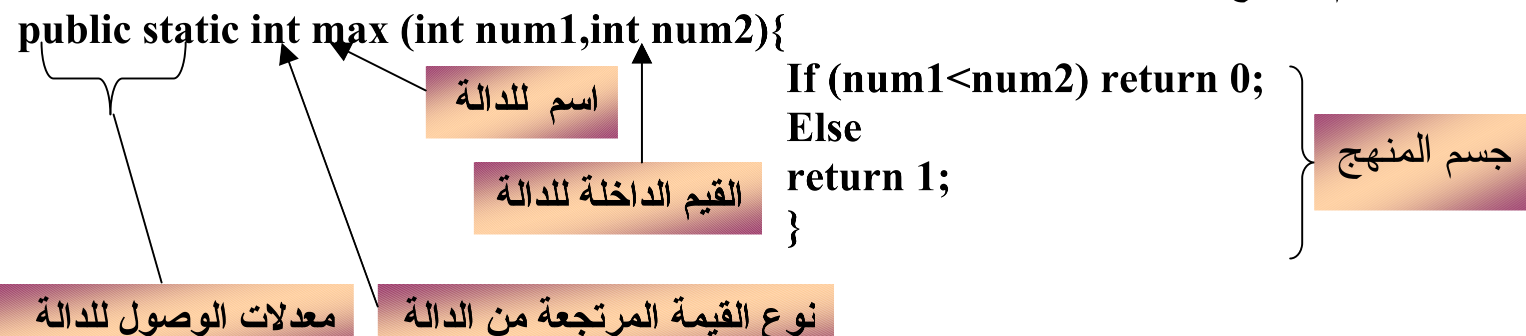
المناهج

- وتسمى الطرق أو الدوال وهى التي من وضع المبرمج
- والهدف منها : انه عند تكرار مجموعة من سطور الأوامر أكثر من مرة في مواضع مختلفة فإن أوامر التكرار لن تكون ذات منفعة . ولذلك يتم كتابة هذه السطور منفصلة عن البرنامج الأساسي

مزايا استخدام الدوال

١- عدم تكرار التعليمات داخل البرنامج : حيث يتم إنشاء الدالة مرة واحدة ثم يتم استدعائها أكثر من مرة عند الحاجة إليها .
٢- باستخدام الدوال يصبح البرنامج أكثر وضوحاً

الشكل العام للمنهج



انواع معدلات الوصول

- public / وهي عامة أي تستطيع الوصول إليها من خارج الكلاس ومن خارج البرنامج أيضا بواسطة الحزم أو الواجهات وهذه مواضيع سيتم شرحها بالتفصيل في الفصول القادمة
- private / أي بمعنى خاصة فتستطيع الوصول للداة من داخل الكلاس فقط ولايمكن ان توصل لها من خارج الكلاس اطلاقاً الا عن طريق حيلة سيتم ذكرها لاحقاً .
- protected / أي بمعنى محمي أي انك تستطيع الوصول للدالة من داخل الكلاس لو من خارج الكلاس اذا كان الكلاس يرث منة وسيتم ذكرها في فصل الوراثة .

استخدمنا العبارة static من اجل اخبار المترجم على ان هذة الدالة من نوع ثابت أي انة يقوم بالتعرف عليها قبل الدخول الى الدالة الرئيسية .
وهذا كود بسيط يبين عمل الدوال ويقوم بتربيع عدد ما

```
class bv {
        public static void t (int b){
                                System.out.println(b*b);
                               }

public static void main(String[] args) {
```

t(3);
}
}

أنواع الطرق :

نستطيع تقسيم الطرق إلى نوعين حسب الإعادة . فبعض الطرق التي يتم تحديد نوع القيمة المرجعة، تقوم بإرجاع قيمة عن طريق استخدام الكلمة return . بينما لا ترجع الطرق من نوع void شيئاً.

```
public void setDate(int d, int m, int y){
    day = d;
    month = m;
    year = y;
}
```

```
public String getDate(){
    String str=day+"/"+month+"/"+year;
    return str;
}
```

كما يمكننا تقسيم الطرق حسب وضعية الوظيفة في الفئة إلى نوعين، طرق خاصة بالفئة، وطرق خاصة بالعضو. و يتم في النوع الأول كتابة كلمة (static) في توقيع الوظيفة (method signature). و هكذا نكون قد جعلنا هذه الوظيفة هي خاصة بالفئة بشكل عام و ليست خاصة لعضو من الأعضاء. و بإمكان أي عضو استخدامها من الفئة مباشرة دون الحاجة لإنشاء عضو من الفئة. و كمثال على ذلك نفس المثال السابق لتربيع العدد .

فنحن نستطيع استخدامها بالإشارة لاسم الفئة مباشرة دون الحاجة لإنشاء عضو في الفئة، و استخدامها عبره . تماماً كما في المثال السابق، يمكننا أن نجعل طرق الجمع static و نستخدمها مباشرة دون الحاجة لإنشاء عضو من فئة AddTwo، هكذا:

```
1  public class AddTwo {
2
3      static int sum(int i, int j){
4
5          return i+j;
6      }
7
8      static float sum(float a, float b){
9
10         return a+b;
11     }
12
13 }
```

نلاحظ أننا عرفنا الفئات لتكون static في السطر الثالث و الثامن، و هكذا نكون قد جعلناها خاصة بالفئة بشكل عام لا بعضو من الأعضاء. و الآن فقط نستطيع أن نستخدمها بهذا الشكل:

```
1  public class AddMain {
2
3      public static void main(String str[]){
4
5          int i1=10, i2=12, intResult;
6          float f1=1.2f, f2=3.49f, floatResult;
7
8          intResult = AddTwo.sum(i1,i2);
9          floatResult = AddTwo.sum(f1,f2);
10
11         System.out.println(intResult);
12         System.out.println(floatResult);
13
14     }
15 }
```

نلاحظ أننا في السطرين الثامن و التاسع استخدمنا وظيفة sum مسبوقة باسم الفئة AddTwo مباشرة، دون الحاجة لاستخراج عضو من الفئة AddTwo، لأنه تم تعريف الوظيفة على أنـها static.

- واليكم هذا الكود فهو نفس السابق ولكن الاختلاف إننا لم نعرف الدالة من نوع static ولكن مادام إننا لم نعرفها من نوع ستاتيك فعملنا على إطلاق هدف من نفس اسم الكلاس وسيتم التطرق لعملية إطلاق الأهداف لاحقاً المهم أحببت أن أبين لكم الاختلاف بين الكوديين

```
class aldopaee {
public  void t (int b){System.out.println(b*b);}

  public static void main(String[] args) {
      aldopaee b=new aldopaee ();
      b.t(3);}
}
```

- قاعدة / عند كتابة الدالة مثل هذا الكود فيجب إطلاق هدف باسم الكلاس نفسه ومن ثم كتبة اسم الهدف الجديد.اسم الدالة المراد استخدامها وان كتبت اسم الدالة مباشرة فان المترجم سيصدر خطاء ولن ينفد البرنـامج .

وهذا الكود لن ينفد إطلاقا

```
class aldopaee {
public  void t (int b){System.out.println(b*b);}

  public static void main(String[] args) {t(3);}
}
```

- من الممكن استخدام الجملة return مع الطريقة التي لا تعيد قيمة والغرض منـها إيقاف الطريقة في أي لحظة وهذا الكود يبين ذلك .

```
class aldopaee {
    static void b(){
                    if(true){System.out.println("1");return;}
                    System.out.println("2");
                   }
public static void main(String args[]){
b();
System.out.println("3");
}
}
```

```
D:\JCreato
1
3
Press any
```

❖ هل هذا الكود صحيح ؟

```
class aldopaee {
    static void b(byte a , int b){  System.out.println(a+b);}
```

```
public static void main(String args[]){
b(1,200);
}
}
```

طبعاً عند النظر إليه للوهلة الأولى سنقول أنة صحيح ولكن عند تنفيذه سنواجه خطاء فما هو هذا الخطأ وكيف تعالج هذا الخطاء ؟

❖ لديك طريقتين لتصحيح هذا الخطاء حاول اكتشافهما !

التحميل الزائد للمناهج

هو عبارة عن دالتين أو أكثر تحمل نفس الاسم إلا إنها تختلف في عدد الوسائط أو الأنماط .
أي مثلاً أنت عندك ثلاثة مسدسات ربع ونص وكامل فكل مسدس لدية رصاص خاص به وكلهم يحملان نفس الاسم ويميز الأول عن الثاني عن الثالث بالرصاص وحجمهم فقط وعند تسليمك رصاص من نوع صغير أنت تلقائياً ستفهم أن هذا الرصاص خاص بالمسدس الربع وستقوم بشحن الرصاص بالمسدس .
هذا هو المترجم عندما يكون في البرنامج عدة دوال بنفس الاسم الأولى تأخذ من نوع أنتجر والثانية تأخذ من نوع تشار والثالثة من نوع فلوت فأنة عند إرسال قيمة من نوع أنتجر فان المترجم سيستخدم الدالة التي من نفس نوع القيمة المرسلة .

```
class Overload{
 public static void main(String args[]){
 sum();
 sum(100,3);
 System.out.println("sum= " + sum(8.5, 4));
 System.out.println("sum= " + sum(10, 4.2));
 System.out.println("sum= " + sum(8, 9, 4));
 }
 static void sum(){int num1 = 10, num2 = 5;System.out.println("sum = " +
(num1 + num2));}
 static void sum(int num1, int num2){System.out.println("sum = " + (num1 +
num2));}
 static double sum(double num1, int num2){return (double)(num1 + num2);}
 static double sum(int num2 ,double num1){return (double)(num1 + num2);}
 static int sum(int num1, int num2, int num3){return num1+num2+num3;}
}
```

```
D:\JCreator LE\GE2001.exe
sum = 15
sum = 103
sum= 12.5
sum= 14.2
sum= 21
```

ملاحظات حول التحميل الزائد

- تجعل البرنامج واضحاً وأكثر قابلية للقراءة ويجب إعطاء نفس الاسم للدوال التي تقوم بانجاز عملها .
- لا يمكن الحصول على دوال ذات تحميل زائد من خلال نمط القيمة المرتجعة من الدالة فهذا خطاء شائعاً.

- في بعض الأحيان قد يحصل توافق يالبرمترات عند الدوال ذات التحميل الزائد فيفشل المترجم اختيار الدالة المناسبة بما معنى انه غموض فهذا الكود خاطئ

```
class Overload{
 public static void main(String args[]){
 System.out.println(max(3.1,10));
 }
static void double max(double num1, int num2){
   if(num1>num2)return num1;else  return num2;}

static void double max(int num1,double num2){
   if(num1>num2)return num1;    else   return num2;}
}
```

يمكن اختيار max(int num1,double num2) او max(double num1, int num2) ليتوافق مع max(3.1,10) حيث لا يوجد منهم محدداً أكثر من الأخر وبالتالي هذا الاستدعاء غامض .

البرمجة الهدفيه

مقدمة : -

ماذا نعني بالبرمجة الكائنية ؟

OOP(Object Orientation Programming) البرمجة كائنيه المنحى " البرمجة الموجهة بالكائنات " هي ذلك المفهوم الذي بزغ إلى عالم البرمجة ليغير طريقة البرمجة الإجرائية القديمة ذات الدوال والمناهج الضيقة الأفق إلى سعة ورحابة الكائنات

منذ بزوغ فجر تاريخ البرمجة بدأ المبرمجون بكتابة برامج باستخدام لغة الآلة فوجدو بعد فترة أن هذه اللغة متعبة لحد يجعل تطور البشرية في هذا المجال أمرا صعبا للغاية فقرروا تطوير البرمجة لاستخدام دلالات تعبر بلغة أقرب للغة البشر عن برمجة الصفر والواحد فاخترعوا لغة الاسمبلي " التي ليست إلا اختصارا لتعليمات ست عشرية هي في الأصل صفر وواحد " فتطورت البرمجة بشكل كبير وسريع لكن مع ازدياد الحاجة البشرية للسرعة قرروه أن هذه اللغة تأخذ وقتا طويلا للغاية فقرروا مرة أخرى تبسيطها أكثر فبدأو باختراع لغات البرمجة عالية المستوى أمثال الكوبول والباسكال والبيسك وأمثالها الكثير وكانت كلها مسيطرة في وقتها حتى بزغ فجر لغة السي

مرة أخيره " حتى ألان " قرر المبرمجون أنهم بحاجة إلى التبسيط زيادة فقرروا أن يحاولوا محاكاة الواقع فلا أسهل من التعامل بشكل طبيعي مع الأشياء وهنا بزغ فجر لغات البرمجة كائنيه المنحى

ألان عندما أقول كائن هذا المصطلح غامض بعض الشيء لكن يمكنك تشبيهه فورا بمفهوم الكائنات التي في العالم الحقيقي يمكن أن يكون الكائن إنسانا حيوان جماد مثل المكتب المصعد الكهربائي وحتى كرة القدم

ألان ما العلاقة بين كائنات العالم الحقيقي وكائنات البرمجة ؟؟

عندما فكر مخترعو البرمجة الكائنة بهذا المفهوم الجديد كل ما كان لديهم في ذلك الوقت هو تسهيل البرمجة بأكبر فرصة لتصبح مشابهة للتصرفات على الواقع تماما

فكر المخترعون على طريقة لإبعاد المبرمج كليا عن طريقة عمل كائن ما في البرمجة بحيث يركز عمله فقط على كيفية استعماله !!

لتركيز هذا المفهوم في الواقع خذ عندك مثالا : لعبة رجل إلي يلعب بها طفل ويحركها بيديه ويضغط فيها أزرارا لتصدر بعض الأصوات والحركات وتنفذ بطاريتها فتتوقف عن اللعب ويرميها في الأرض فتتحطم !!

ألان هذا الطفل لن يعرف مطلقا كيف يعمل هذا الرجل الآلي كيف يتحرك إذا ضغطنا هذا الزر كيف يصدر صوتا إذا ضغطنا ذلك الزر !!

هذا مشابه تماما لما يريدنا مخترعو ال OOP الوصول إليه أن نتحكم بالكائنات بكل سهولة دون الدخول في تفاصيل طريقة عملها

ومن هنا بزغ فجر مفهومين جديدين للبرمجة " صانعو الفئات " "ومستخدمو الفئات "

صناع الفئات هم كما في لعبة الرجل الآلي الشركة المصنعة لهذه اللعبة

والمستخدمون هم الأطفال الذين يلعبون بها ولا يعلمون شيئا عن طريقة عملها الداخلية فقط يصدر الصانعون Manual لطريقة الاستخدام لكي يعرف الطفل كيف يستمتع بها وهو تماما ما يحدث في كائنات ال OOP

ألان هل يمكن فعلا أن تكون البرمجة بهذه السهولة ؟ أقول نعم إذا ركز كل على عمله

مصنعو الفئات سيكون بالطبع عليهم العبء الأكبر المستخدمون قد يكون عليهم عبء وقد يكونون في قمة حالات الاستمتاع بهذا الكائن

حالات الاستمتاع في الواقع كما لدينا الطفل الذي يلعب بالكائن الآلي الكامل وهي أخر مراحل استخدام الكائن لأن الطفل لن يستخدم الكائن ليطوره لكائن اخر "إلا إذا كنا في عالم ال Matrix ونحن لا نعلم !!!" فقط سيكتفي باللعب به

أما لوكنا في مثال اخر لو كان الكائن الحالي لدينا هو عبارة عن محرك سيكون هناك بعض العبء على

مستخدم الكائن الذي سيقوم بتركيبه مع عدة كائنات أخرى ليكون في النهاية كائنا جديدا ... هنا نحن لم ننتهي من سلسلة التطوير لهذا الكائن بعد فيمكن اعتبار المستخدمين مطورين

بهذا المفهوم مطورو المحركات سيبيعونها لمصنعين أخريين وبهذا التكامل نبني واقعنا في الحياة

نفس المفهوم تماما موجود في عالم البرمجة OOP لكن من يستطيع الوصول لهذه المراحل من التطوير من قال أنه لا يوجد لو دخلت ورأيت برمجة الألعاب ستجد العجب العجاب

ولو اضطلعت على نماذج محاكاة الواقع الافتراضي فهي القمة في استخدام الكائنات لأنها تبنى أساسا على محاولة محاكاة كائن في الطبيعة بشكل حقيقي تماما ليعمل على الكمبيوتر بنفس طريقة عمله في الطبيعة مثلا متتابعات الأشعة ومحاكاة حركة الرياح والأعاصير محاكيات أحوال الطقس محاكيات التفاعلات الكيميائية وغيرها

مثلا في محاكيات التفاعلات الكيميائية سيكون المطورون بداية كائن هو عبارة عن ذرة بالكتروناتها ونواتها وبوزوتروناتها وبروتوناتها وكل محتوياتها

هذا الكائن سيدمج في كائن أكبر منه وهو الجزيء سيتكون من عدة كائنات ذرة ثم نتدرج حتى نصل إلى المادة الكيميائية ويكون مبرمجو الكائنات السفلية قد اطلعوا على كيفية تفاعل الجزيئات مع بعضها بشكل تام ثم يبدؤون بكتابة " الدوال (أقول الدوال هنا وأنا لا أمزح) " التي ستقوم بعملية الالتحام الدمج بين الجزيئات ويملأنها بكل تفاصيل التفاعلات في النهاية فقط ما على مستخدم الكائن النهائي وهو " كائن بيانات المحلول " إن ندخل له بيانات المحلول الأول والثاني ونطلب منه أن يفاعل بينهما وننتظر نتيجة التفاعل !!!!

هذه الاشياء بالطبع تحتاج لكمبيوترات عملاقة سريعة لتنفذ كل هذا الكم من التعليمات

لكن يمكن التدرج وصولا لمستويات مبرمجي الالعاب حيث يقومون ببناء ألعابهم على أساس الكائنات مثلا خذ عندك لعبة بلياردو وهو مثال أوضح نوعا ما

ماعلى مطوري اللعبة إلا استخدام كائنات كرة بلياردو " لأنها الجزء الأصعب " كائن البلياردو هذا سيتعامل كما في الحياة الواقعية تماما سيكون الكائن عبارة عن جسم كروي له كتلة بافتراض ان الجاذبيه الارضية ٩,٨ سنعطيه أيضا مكان لتخزين معلومات طاقته الحركية وطاقته الكامنة فكل ماعلينا هو كتابة دالة لتقوم بعملية التصادم بحيث أن كل كرة عندما تصطدم بكرة أخرى ستستمد طاقة حركية وطاقة كامنة داخلها بهذا المبدأ يمكن أن نحرك كراتنا وننسى تماما كيفية تصادمها وانعكاسها !!!

أردت أن أبين هذه الأبور لأنها الأشياء التي أتعبتني في فهم الكائنات بالشكل الصحيح لم لأجد كتابا يتحدث

عنها بالشكل المفروض كل الكتب تعطي أمثلة سطحية سريعة مباشرة لا تعبر عن الاستخدام الأمثل للكائنات فمثلا لو قلت لك مثال مصعد كهربائي هو عبارة عن كائن ستقول لي يمكن أكتب برنامجا كهذا دون الدخول في تفاصيل الكائنات باستخدام لغة إجرائية بسيطة !! فيصبح المبرمج المبتدئ الذي سيكون ضيق الأفق في البداية مشوشا لا يعرف الاستخدام الأمثل لهذه التقنية

أخيرا قبل أن أنتهي من هذه المقدمة الفلسفية أقول أن المستقبل سيحمل فقط لغات كائنيه المنحى من لم يرد الدخول في ذلك سيسقط وما عليه إلا بانتظار قدره

فأمر الكائنات ليس معقدا بل مفهومه مختلف فقط و لنفرض أن لدينا مبرمج يريد انشاء لعبة تصويب ثلاثية الأبعاد مثل Quake3 مثلا

بها شخصيات وأناس يتحركون ويتصرفون بشكل ذكي وكأن لهم عقول يفكرون بها

ألان انظر إلى حال أحد المبرمجين القابعين أمام أجهزتهم كل الوقت وهو يكتب كود تكامل اللعبة مع بعضها لولا الكائنات في البرمجة لظل هذا المبرمج ٦ سنوات وهو يحاول أن يكامل بين ألاف الأجزاء في مقابل أن يجلس سنتين فقط وهو يستعمل كود للكائنات

"الألعاب الكبيرة تستغرق فترة متوسطها سنتين "

ألان لنفرض مثلا أن هذا المبرمج لا يستخدم كود كائني سيضطر في كل frame أن يتكفل بتحريك كل شخصية في اللعبة ويقلق بشأن تصرفها هل هو سليم أم لا هل تعدى الكائن الفلاني حدود المشهد أم لا هل اصطدم شخصين مع بعضهما في المشهد أم لا سيجن جنونه وهو يحاول ملاحقة هذه الاحتمالات وكل تعديل طفيف سيأخذ منه وقتا كبيرا وكل تعديل كبير يمكن أن يؤدي بالمشروع إلى الهاوية

هذا بالنسبة لحال مبرمج واحد فما بالك إذا تشارك فريق لتطوير اللعبة يجب عليهم أولا أن يتواجدوا في مكان واحد واحتمال تضارب الاكواد بينهم كبير لدرجة تجعل من المستحيل تنفيذ المشروع

في المقابل افرض أن مبرمجنا يستخدم كود كائني المنحى سيتم تقسيم أعضاء المشروع إلى فرق كل فريق له مهمه واضحة محددة كالشمس مثلا الفريق الذي سيهتم بكتابة كود الشخصيات سيقوم بكتابة فئة تعرف الشخصية ويضع كل الاحتمالات الممكنة لهذه الشخصية الحركة التخاطب الأصوات حدود المشهد التصادم بين الشخصيات ماذا لو اصطدمت الشخصية بأخرى قد ترتد وتصدر صوتا مثلا أو غيرها من الاستجابات حتى الان هذا الكائن بدأ يتجسد بالطبع بعد مكاملة فريق رسم الشخصية مع المبرمج يبقى أمر مهم بث

الحياة في هذه الشخصية !! كيف يمكن بث الحياة فيها

بعد تعريف الفئة وتعريف كل المتغيرات الضرورية فيها والدوال التي ستقوم بالأبور المهمة تبقى الدالة الأب التي هي في الواقع كأنها العقل البشري الذي يحدد ما يجب فعله حسب التغيرات الخارجية " لا يمكن بالطبع جعلها تتصرف كالعقل البشري " مثلا لنفرض أن الشخصية ستكون حارس لبوابة وكل من يقترب من هذه البوابة سيتم التصدي له

سنكتب دالة اسمها UpDate يتم استدعاءها كل Frame مثلا بحيث يتم مسح دائرة نصف قطرها ٨ أمتار من الشخصية وإذا وجدت شخصية أخرى في هذا المدى تستدعى دالة أخرى لتحفيز القتال !!! دالة تحفيز القتال ستستدعي دالة لتغير وضعية الشخصية الرسومية ثم تستدعي دالة الهجوم وهكذا بسلسلة كهذه من الاحتمالات الأساسية يكون لدينا في النهاية مقاتل صنديد يتصرف بتلقائية وبالشكل المطلوب

ألان فلنعد لمبرمجنا الذي كان سيقضي ٦ سنوات وكأنه يقضيها في السجن ونعطيه فئة الشخصية وأنواع الشخصيات الاخرى سيكون سعيدا جدا لأنه لن يفعل شيئا في كل Frame إلا أنه سيستدعي الدالة UpDate كل مرة وينتهي الأبر !!! لأن الدالة هي التي ستجعل الكائن يتصرف هكذا يمكن لكل عضو في الفريق أن يركز فقط على عمله وبشكل مدهش وأن يعملوا مع بعضهم بشكل فعال حتى لو كان بينهم ألاف الأميال !!

ألان هذه الفئة فئة الشخصية حجمها قد يكون كبير لكن مبرمج اللعبة لن يقلق بشأنها فليس له أي علاقة بحجمها فقط كل ما عليه هو أن يضعها ويقرأ طريقة استخدامها وينسى كل شيء ويعتمد على أن مبرمج الفئة قد أتقن عمله فعلا

هنا تقريبا يكمن العبء الأكبر على مبرمج الفئة حيث يجب أن يكون حذرا ويتأكد بشكل كبير من عمل الفئة بالشكل الصحيح .

تصنيف الكائنات إلى صنفين:

- كائنات نشطة حية (Animate Objects) وهي التي نحس فيها فنجد لها حركة ونشاط.
- كائنات غير نشطة غير حية (Inanimate Objects) هي التي لا نلاحظ لها نشاط أو حركة أو وقع أينما وجدت .

وجميع الكائنات بصنفيها لها:

١. خصائص Attribute مثل :الحجم، اللون، الوزن، الشكل...ألخ .

٢. سلوك Behavior فمثلاً: الطفل (كائن) يبكي، وينام، ويمشي، ويأكل (سلوكيات) .

الإنسان وخصوصاً المبرمج يتعلم عن الكائنات بمعرفة خصائصها، وملاحظة (تجربة) سلوكها، فمن الممكن أن يكون لكائنات مختلفة نفس الخصائص وسلوك متقارب.

لماذا الكائنات مهمة جدا ؟

هناك الكثير من الأسباب ، دعني أعطيك بعضاها :

١- قدرتك على معرفة مكان الخطأ بسهولة إذا حصل

٢- القدرة على تتطوير البرنامج بسهولة مع الوقت

٣- القدرة على إعادة استخدام الكثير من أجزاء البرنامج لتطوير برامج أخرى

٤- عدم الحاجة لإعادة كتابة الشفرة البرمجية عند كل إصدار جديد للبرنامج

٥- سهولة تحويل الشفرة البرمجية للغة مختلفة

٦- القدرة على توزيع العمل في برنامج واحد ضخم على أكثر من مبرمج بسهولة ويسر.

فوائد البرمجة بالأهداف

- ☒ حماية البيانات فكما قلنا أن المبرمج الذي يستخدم الفصيلة لا يرى غالباً الكود المبني بة هذه الفصيلة ولكنة فقط يتعامل مع الدوال والبيانات الموجودة التي تعرضها الكلاس حتى لا يستطيع احد التغير ولو عن طريق الخطاء .
- ☒ الكبسلة .
- ☒ الوراثة .
- ☒ تعدد الأشكال .

إنشاء الأهداف

لنفترض أنه عندنا class أسمه مصباح Light وال methods التي تقدمها هي ينير on و يغلق off

فكيف نعبر عن هذا في الجافا

```
public class Light {
public static void main ( String [] arge) {
Mesbah aMesbah = new Mesbah ();
```

```
aMesbah.on();
aMesbah.off();
}
}
```

فهنا قمنا بتعريف object أسمة mesbah (مصباح) من ال class التي أسمها Light، ولكي يتم تخليق هذا المتغير فاستخدمنا ال مصطلح new ثم كتبنا أسم ال class ثم ننهي الأمر باستخدام فصلة منقوطة.
ولكي نقوم بطلب خدمة on أي يضيء على المصباح فقد كتبنا اسم الobject مصباح ثم نضع نقطة (.) ثم نكتب أسم ال method (الخدمة) التي نريدها كما في السطر الثاني.

استخدامات الكلمة ألمفتاحيه (new)

١) يتم إنشاء هدف من الفصيلة المعلن عنها .

٢) يتم حجز جزء من الذاكرة لهذا الهدف .

٣) يتم استدعاء دالة البناء الخاصة بهذه الفصيلة .

دوال البناء Constructor

وهو عبارة عن طريقة التكوين التي يتم بها إنشاء العضو من الفصيلة فتأخذ نفس اسم الفصيلة وتنفذ عند إنشاء الفصيلة و من التنويهات لها

- **تستطيع إعطاء قيم ابتدائية لمتغيرات فصيلة الهدف .**
- **يجب أن تملك دالة البناء نفس اسم الكلاس .**
- **لا تملك البانيات نمط إرجاع ولا حتى void .**
- **يتم استدعاء البانيات باستخدام العامل new عند إنشاء الكائن .**
- **إذا لم يعرف الكلاس الرئيسي أي باني بشكل صريح فأنة تلقائياً يتم تعريف باني افتراضي وهذا الكود يبين ذلك.**

```
class Bird {int i;}
public class aldopaee {
  public static void main(String[] args) {
```

```
        Bird nc = new Bird();
      }
    }
```

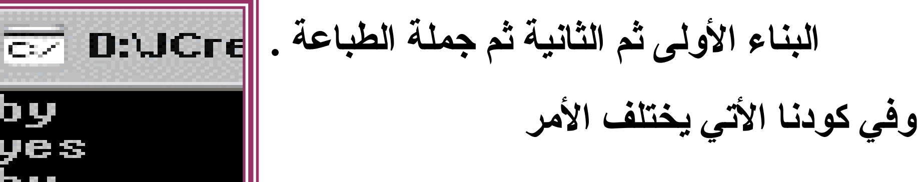
- ☒ من الأخطاء المرتكبة وضع كلمة void or int قل اسم الباني فيصبح طريقة وليس بانياً
- ☒ يمكن تعريف ال Constructor بحيث يكون فارغاً من الكود لأسباب .
- ☒ إذا قمت بتعريف constructor خاص بك، فأنك تفقد الـ constructor الافتراضي، فإذا أردت أن تحتفظ به، عليك أن تقوم بكتابته يدوياً .
- ☒ يمكن أن يكون لنفس الفئة أكثر من Constructor يختلفون في أعداد أو أنواع المتغيرات في سلسلة المتغيرات الممررة لهم، أو كلاهما .
- ☒ شكل الـ Constructor قد يشبه شكل الوظيفة، و لكن تذكّروا دائماً. اسم الـ Constructor هو نفس اسم الفئة، و لا يوجد له نوع بعكس الوظيفة .
- ☒ إن عمل new هو إنشاء العضو من الـ constructor المناسب. فإذا لم يوجد constructor في الفئة تقوم new باستخدام الـ constructor الافتراضي .

```
class aldopaee {

public static void main(String args[]){
        new aldopaee();
        System.out.println("by");
        new aldopaee(10);
}
aldopaee( ){ System.out.println("yes");}
aldopaee( int b){      System.out.println(b+b);}
}
```

```
D:\JCreator
yes
by
20
Press any k
```

- نلاحظ من كودنا السابق انه عندما تريد تفعيل دوال بناء الكلاس نفسه يتم بـ new تم اسم دالة البناء أو بإنشاء هدف من نوع الكلاس aldopaee m=new aldopaee(); .
- نستنتج أيضا أن دوال البناء تسبق أي جمل مكتوبة بداخل الكود فكان تنفيذ البرنامج أولا ينفذ ذاله البناء الأولى ثم الثانية ثم جملة الطباعة .



وفي كودنا الآتي يختلف الأمر

```
class aldopaee {
        {System.out.println("by");}
public static void main(String args[]){
        new aldopaee();
   new aldopaee(10);
}
aldopaee( ){   System.out.println("yes");}
aldopaee( int b){      System.out.println(b+b);}
}
```

- نلاحظ أنة تم تنفيذ جملة الطباعة التي خارج الدالة الرئيسية التي بداخل البلوكات المسماة الكتل المعشعشة مرتان مرة عند تنفيذ دالة البناء الأولى ومرة عند تنفيذ دالة البناء الثانية ،
- ومن الملاحظ أن في الكود السابق كانت دالة البناء تسبق أي جمل مكتوبة ولا كن هنا الأمر يختلف ألان فأي جمل مكتوبة بداخل بلوكات وموقعها خارج الدالة الرئيسية فإنها تسبق تنفيذ دالة البناء .
- جملة الطباعة السابقة لا تنفذ إطلاقا إلى إذا حدث إطلاق للكلاس

```
class aldopaee {
    {System.out.println("by");}
public static void main(String args[]){
   System.out.println("start");
}
}
```



- في هذا الكود أيضا يختلف الأمر

```
class aldopaee {
public static void main(String args[]){
    new aldopaee();
}
aldopaee( ){      System.out.println("yes");}
              {System.out.println("by");}
        static{System.out.println("start");}
}
```



- نلاحظ انه نفذ محتوى static ثم محتوى الكتلة المعشعشة ثم دالة البناء .

- ونستنتج أن دائماً يتم تنفيذ محتوى static قبل كل شيء حتى وان لم يتم إطلاق الكلاس

```
class aldopaee {
public static void main(String args[]){
System.out.println("by");
}
    static{System.out.println("start");}
}
```

```
D:\J
start
by
```

- في هذا الكود قاعدة مهمة جداً

```
class ammar{ int i ; ammar(int a){i=a;}}
class aldopaee {
public static void main(String args[]){
ammar a=new ammar(10);
ammar a1=new ammar(100);
System.out.println(a==a1);

a=new ammar(100);
System.out.println(a==a1);

a=a1;
System.out.println(a==a1);

a.i=1000;
System.out.println(a1.i);
}
}
```

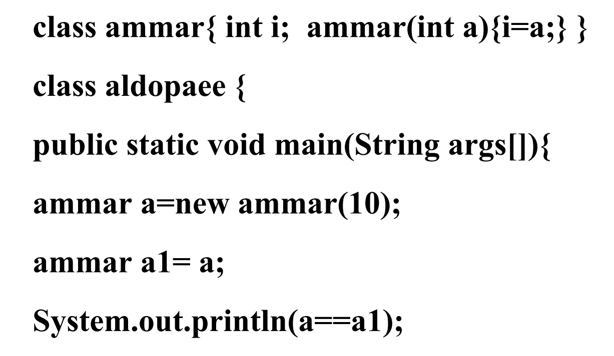
```
D:\JCre
false
false
true
1000
```

هنا ليس بمعنى أن محتوى الكلاس الأول يساوي محتوى الكلاس الثاني لا ولكن هل موقع الكلاس الأول يساوي موقع الكلاس الثاني في الذاكرة .

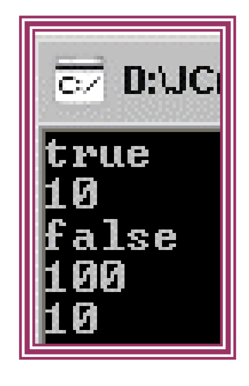
رغم إننا ساوينا محتوى الكلاسين إلا إننا نجد ناتج أمر الطباعة false والسبب ذكرناه سابقاً

السؤال هنا كيف تغيرت قيمة a1.i ونحن غيرنا قيمة a.i الإجابة هو عندما عملنا a =a1 فأنة أصبح a يؤشر إلى موقع a1 وأي تغير في a.i تتغير قيمة a1.i والعكس .

- وهذا الكود أيضا به قاعدة



```
class ammar{ int i;  ammar(int a){i=a;} }
class aldopaee {
public static void main(String args[]){
ammar a=new ammar(10);
ammar a1= a;
System.out.println(a==a1);
```



هنا تم إعادة إطلاق الهدف وإرسال ل I القيمة 100 بما معنى أن موقعة ألذاكري قد تغيـر

```
System.out.println(a1.i);

a=new ammar(100);

System.out.println(a==a1);
System.out.println(a.i);
System.out.println(a1.i);
}
}
```

مصفوفة الفئات

والقصد هنا تكوين عدة فئات من كلاس واحد وتتم بخطوتين

١) تعريف عدد الفئات المراد تكوينها

٢) اشتقاق الفئات

```
class ammar {static int count;
             int j;
           ammar() { j=count;count++;}
         }
class aldopaee {
   public static void main(String[] args) {
      int i=10,j;
     ammar p[] =new ammar[i];
     for(j=0;j<i;j++) p[j]=new ammar();

     System.out.println("j"+" "+"count");
     for(j=0;j<i;j++) System.out.println(p[j].j+" "+p[j].count);
 }
}
```

```
D:\JCre
j count
0 10
1 10
2 10
3 10
4 10
5 10
6 10
7 10
8 10
9 10
```

استخدامات الكلمة ألمفتاحيه (this) / له دور كبير مع دوال البناء وعموماً هذه الكلمة تشير إلى الكلاس نفسه ولها استخدامات .



❖ تستدعي دوال البناء .

```
class ammar{
      ammar(){System.out.println("start");}
    ammar(int b){this();System.out.println(b*b);}
          }
class aldopaee {
public static void main(String args[]){
      ammar b=new ammar(100);
}
}
```

- عند استدعاء باني يجب وضع الكلمة ألمفتاحيه **this** قبل أي جملة وإلا المترجم سيصدر خطاء وتعتبر من الأخطاء الشائعة فهذا الكود لن ينفد إطلاقا .

```
class ammar{
      ammar(){System.out.println("start");}
    ammar(int b){System.out.println(b*b);this();}
          }
class aldopaee {
public static void main(String args[]){
      ammar b=new ammar(100);
}
}
```

❖ تحل مشكلة أسماء الأعضاء المتشابهة .

في بعض الأحيان قد يتشابه اسم البارمتر المرسل للدالة واسم المتغير في داخل الكلاس ..

```
class ammar{int b;
      ammar(int b){this.b=b;}
              }
class aldopaee {
public static void main(String args[]){
   ammar a= new ammar(10);
   System.out.println(a.b);
```

```
}
}
```

٣- تعيد قيمة من نوع الكلاس نفسه مثل الدالة عندما تعيد قيمة وهذا الكود يبين ذلك

```
class ammar{int b=0;
   ammar (){}
   ammar a(){b++;return this;}
   void print(){System.out.println(b);}
        }
class aldopaee {
public static void main(String args[]){
   ammar q=new ammar();
   q.a().a().a().a().a().print();
}
}
```

C:/
5

- لا نستطيع استخدام الكلمة **this** في استدعاء أكثر من دالة بناء .

مجال تغطية المتغيرات بشكل أوسع

واليكم هذا الكود الذي يبين مجال تغطية المتغيرات

```
class aldopaee {
  static int i=10;
   static void print(int i){System.out.println(i);
                               System.out.println(aldopaee.i);
                              }
public static void main(String args[]){
  int i=100;
  System.out.println(aldopaee.i);
  System.out.println(i);
  print(1000);
}
}
```

C:/ D:\
10
100
1000
10

اعتقد من خلال الشكل يتضح شرح البرنامج .

- هذا الكود خاطئ لماذا ؟

```
class aldopaee {
    int i=10;
public static void main(String args[]){
    int j=i;
    System.out.println(j);
    System.out.println(i);
}
}
```

وهو استخدام متغير محجوب عن المترجم فلن يقدر أن يصل إلى I إلا إذا أطلقنا هدف من نوع الكلاس أو نجعل I من نوع static

- لا نستطيع أن نعرف متغير من نوع static داخل الدالة الرئيسية فهذا الكود لن ينفذ إطلاقا .

```
class aldopaee {
public static void main(String args[]){
    static int i;
    System.out.println(i);
}
}
```

- من الأخطاء الشائعة استخدام قيمة متغير عادي الى متغير من نوع static فهذا الكود خاطئ

```
class aldopaee {
    int a=b;
    static int b=a;
public static void main(String args[]){
    System.out.println(b);
}
}
```

- من الأخطاء الشائعة إعادة قيمة من نوع عادي بواسطة دالة من نوع static مثل هذا الكود

```
class aldopaee {
    int a=10;
    static int b(){return a;}
public static void main(String args[]){
    aldopaee c =new aldopaee();
    System.out.println(c.b());
}
```

فتصحيح هذا الكود بطرقتين مختلفتين

☒ جعل a من نوع static int a .

☒ إبعاد جملة static من الدالة .

```
}
```

- قاعدة / عند التصريح على متغير من نوع static داخل الكلاس فهذا يعني أن هذا المتغير مشترك لجميع الفئات المشتقة من الكلاس و هذا الكود يشرح ذلك

```
class ammar{static int i;
            int j;
            ammar(int i,int j){this.i=i;this.j=j;}
           }
class aldopaee {
public static void main(String args[]){

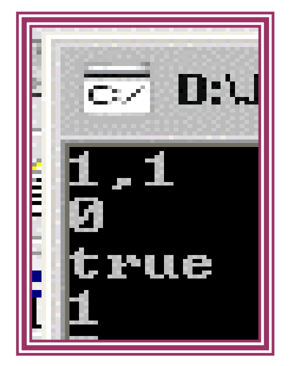
    ammar c =new ammar(10,20),d=new ammar(30,40);
    System.out.println(c.i+" "+c.j);
    System.out.println(d.i+" "+d.j);
}
}
```

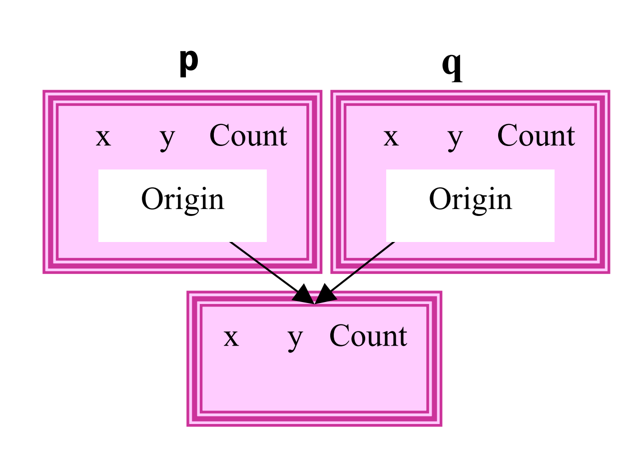
```
30 20
30 40
```

نلاحظ انه عندما أرسلنا القيمة ٣٠ إلى I للفئة d فان قيمة I قد تغيرت من ٢٠ إلى ٣٠ تبعاً للقيمة الأخيرة فهذا يعني أن أي تغير للمتغيرات من نوع static فأنة تتغير جميع متغيرات الفئات التي من نوع static .

- إليكم هذا الكود الغريب نوعاً ما

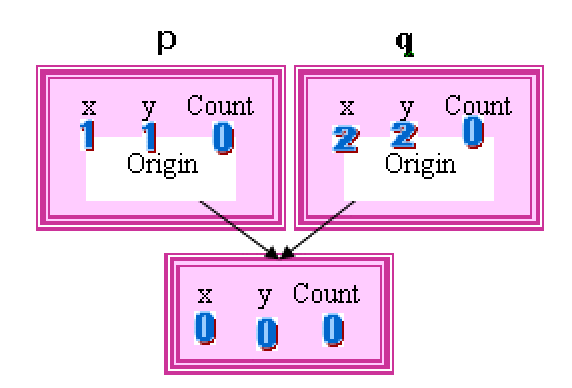
```
class ammar {
    int x, y, Count;
    ammar(int x, int y) { this.x = x; this.y = y; }
    static ammar origin = new ammar(0, 0);
                }
class aldopaee {
    public static void main(String[] args) {
        ammar p = new ammar(1,1);
        ammar q = new ammar(2,2);
        p.Count++; p.origin.Count++;
        System.out.println( p.x + "," + p.y );
        System.out.println(q.Count);
        System.out.println(q.origin == ammar.origin);
        System.out.println(q.origin.Count);
}
}
```



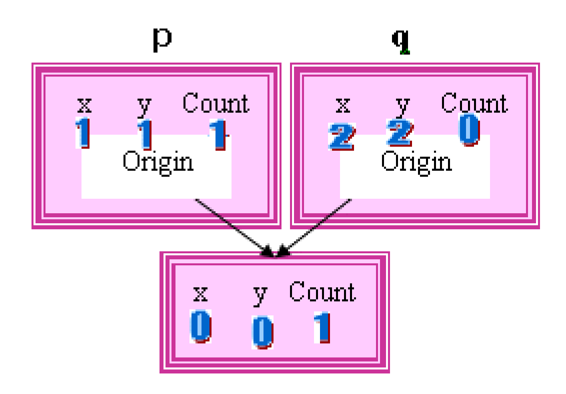


عند تتبع أي برنامج فعليك أن ترسم هيكل الكلاس بهذا الشكل لكي يسهل عليك إيجاد خرج البرنامج

فمن الملاحظ إننا لدينا فئتان q,p وتم إسناد لهم (١،١) و (٢،٢) وداخل كل فئة فئة داخلية لها هيكل بنفس هيكل الفئة الخارجية فبعد اشتقاق الفئات وإرسال القيم لها تكون الفئات بهذا الشكل



p.Count++; p.origin.Count++;



ومن خلال الإشكال السابقة يتضح خرج البرنامج .

معدلات الرؤية والوصول

لقد خلق المبدع والأول والآخر سبحانه الclass الإنسان وجعل له صفات (members = variables) من عينين وأنف وأذن و قلب و الكثير من الصفات الأخرى وقد جعل الله هذه الصفات من الممكن أن نصل لها ونمسكها فنستطيع أن نمسك أعيننا و هناك الطبيب الجراح الذي يمسك القلب بيديه فهذه الصفات يستطيع أن يصل لها أي شيء فهي عامة (public) ، وخلق الله أيضاً صفات (members) في الإنسان مثل الروح ولكننا لا نستطيع أن نصل لها و نمسك بها. يقول تعالى (و يسألونك عن الروح قل الروح من أمر ربي) فهي صفات خاصة (private) ممنوع أن تصل لها أي (class) أخرى . و تخيل لو كانت الروح من الممكن أن نصل لها ونمسكها مثل العين فكما إنه هناك من تمرض عينه فيقوم بنقل عين إنسان آخر فلو كانت الروح من الممكن أن نصل لها (public) لوجدنا من لا تعجبه روحه فينقل ويستبدل روح إنسان آخر ! إذن فهناك)

members) في ال class تكون public وأخرى private لا تستطيع class أخرى أن تصل لها مباشرة. ولكن قد يقوم صانع ال class بعمل methods تؤثر في هذه ال private members ودون أن تصل لها. فمثلا جعل الله الخالق العظيم methods في الإنسان يستطيع بها أن يؤثر في الروح ومن هذه ال methods يصلي و يزكي ويصوم و....

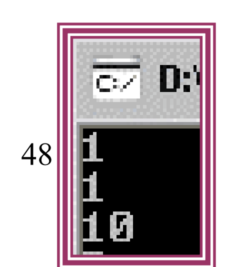
وخلق الخالق سبحانه وتعالى أيضا جينات نمتلكها من أبوينا ولا يستطيع فرد خارج العائلة أن يمتلكها أو أن يحصل عليها مثل النخاع ...

في الجافا يستطيع ال class creator أن يحدد قواعد الوصول (access control) لل members في ال class من variables و methods. هذه access control هي : (public و private و protected و default) .

- Public / وهي عامة أي تستطيع الوصول إليها من خارج الكلاس ومن خارج البرنامج أيضا بواسطة الحزم أو الواجهات وهذه مواضيع سيتم شرحها بالتفصيل في الفصول القادمة
- Private / أي بمعنى مخفية تستطيع الوصول له من داخل الكلاس فقط ولا يمكن أن توصل له من خارج الكلاس إطلاقا إلا عن طريق حيلة سيتم ذكرها لاحقاً وتستخدم في حالات
  ١- عندما تكون الفصائل الأخرى لا تحتاج إلى استخدام تلك المتغيرات .
  ٢- عندما تخشى فصيلة خارجية من ألعبث بالبيانات .
- Protected / أي بمعنى محمي أي انك تستطيع الوصول له من داخل الكلاس أو من خارج الكلاس إذا كان الكلاس يرث منة ولا تستطيع إطلاقا الوصول لهذا النوع من خارج الفصيلة دون وراثة وسيتم ذكرها في فصل الوراثة .
- إذا لم يتم ذكر أي محدد من التي ذكرت فان لغة الجافا تعتبر المتغيرات والدوال و الكلاس عام في داخل الحزمة فقط بخلاف لغة السي تعتبرها من نوع private .

مكان الوصول	public	default	Protected	Private
من داخل الفصيلة	✓	✓	✓	✓
من أي فصيلة داخل الحزمة	✓	✓	✓	✗
من أي فصيلة خارج الحزمة	✓	✗	✗	✗
من أي فصيلة فرعية أي مورثة داخل الحزمة	✓	✓	✓	✗
من أي فصيلة فرعية خارج الحزمة	✓	✗	✓	✗

وهذا الكود يشرح ما سبق



```
class ammar {
      private int x, y;
      public int Count;
      ammar(int x, int y) { this.x = x; this.y = y; }
      int get_x(){return x;}
      private int get_y(){return y;}
      void show(){System.out.println(get_y());}
              }
class aldopaee {
     public static void main(String[] args) {
        ammar p = new ammar(1,10);
        p.Count++;
        //p.x++;//error
        //p.y++;//error
        //System.out.println( p.x + "," + p.y );//error
        System.out.println(p.get_x());
        //System.out.println(p.get_y());//error
        System.out.println(p.Count);
        p.show();
}
}
```

- هذا الكود توجد به داله بناء من نوع محمي ومطلوب منك تنفيذ الكود دون المساس بمحددات الوصول فكيف ستعالج هذه المشكلة ؟

```
class ammar {
      private ammar() { System.out.println("Start");}
           }
class aldopaee {
     public static void main(String[] args) {
        ammar p = new ammar();
        System.out.println("End");

}
}
```

صلي على حبيب خلق الله سيدنا وحبيبنا محمد (اللهم صلي علية وعلى اله وصحبة أجمعين) وركز معي للحظة

لديك طريقتين

١) أن نصنع دالة من نوع static تعيد لنا قيمة من نوع دالة البناء المحمية

```
class ammar {

      private ammar() { System.out.println("Start");}

      static ammar dop(){return new ammar();}

         }

class aldopaee {
```

```
public static void main(String[] args) {

    ammar p =  ammar.dop();
    System.out.println("End");
}
}
```

وعند إنشاء الهدف نذكر اسم الكلاس. اسم الدالة الاصطناعية .

كما نلاحظ أن هذه الطريقة مربكة نوعاً ما وطويلة بعض الشيء .

فإليكم هذه الطريقة التي اكتشفتها من خلال تطبيقي لهذا الكود .

٢) نصنع دالة اصطناعية من نوع static وبداخلها إطلاق لدالة البناء المحمية

```
class ammar {
    private ammar() { System.out.println("Start");}
    static void dop(){ new ammar();}
        }
class aldopaee {
    public static void main(String[] args) {
        ammar p =null;
        p.dop();
        System.out.println("End");
    }
}
```

ثم عند إنشاء الفئة نجعلها تساوي null فبهذا الشكل لن يتم تنفيذ دالة البناء إطلاقا

ثم نذكر اسم الفئة. اسم الدالة الاصطناعية .

بـــــالله علـــــيكم أيهمـــــا أفضـــــل الكـــــود الأول أم الكـــــود الأخـــــر ولمـــــاذا !!

تمرير الفئات إلى الدوال

ونقصد بهذا أن نمرر فئة مشتقة من الكلاس إلى دالة تستقبل فئة وهذا الكود يبين ذلك

```
class ammar {int i;
      ammar() { i=40;}
      void dop1(ammar x){x.i++;}
       }
class aldopaee {
     public static void main(String[] args) {
        ammar p =new ammar();
        ammar q=new ammar();
        p.dop1(q);
        System.out.println(p.i);
        System.out.println(q.i);
        System.out.println(p.i);
}
}
```

```
40
41
40
```

هنا تم إرسال الفئة q إلى دالة dop الموجودة بداخل الكلاس p وتم تغير قيمة I الموجودة بداخل q

أم ألان وبعد أن تعلمنا الأساسيات فما عليك ألان صديقي ألعزيزي إلا أن تركز في هذا الكود فهو مهم جدا

```
class Bowl {
  Bowl(int marker) { System.out.println("Bowl(" + marker + ")"); }
  void f(int marker) {  System.out.println("f(" + marker + ")"); }
}
class Table {
  static Bowl b1 = new Bowl(1);
  Table() { System.out.println("Table()"); b2.f(1);
  }
  void f2(int marker) { System.out.println("f2(" + marker + ")"); }
  static Bowl b2 = new Bowl(2);
}
class Cupboard {
  Bowl b3 = new Bowl(3);
  static Bowl b4 = new Bowl(4);
  Cupboard() { System.out.println("Cupboard()");  b4.f(2); }
  void f3(int marker) { System.out.println("f3(" + marker + ")");}
  static Bowl b5 = new Bowl(5);
}
public class aldopaee {
  public static void main(String[] args) {
```

١) يتم تنفيذ الجمل الاستاتيكية في الدالة الرئيسي .
٢) بداخل كل كلاس كتل استاتيكي يقوم بتنفيذه أولاً ثم الجمل الغير استاتيكية مثل جمل إنشاء الفئات .
٣) يقوم بتنفيذ دوال البناء داخل كل كلاس .
٤) أن عاود استدعاء الكلاس مرة أخرى فأنة لا ينفذ الكتل الاستاتيكية بل ينفذ
١) الكتل العادية .
٢) دوال البناء .

```
    System.out.println("Creating new Cupboard() in main");
    new Cupboard();
    System.out.println( "Creating new Cupboard() in main");
    new Cupboard();
    t2.f2(1);
    t3.f3(1);
  }
  static Table t2 = new Table();
  static Cupboard t3 = new Cupboard();
}
```

```
D:\JCreator LE\GE2001.exe
Bowl(1)
Bowl(2)
Table()
f(1)
Bowl(4)
Bowl(5)
Bowl(3)
Cupboard()
f(2)
Creating new Cupboard() in main
Bowl(3)
Cupboard()
f(2)
Creating new Cupboard() in main
Bowl(3)
Cupboard()
f(2)
f2(1)
f3(1)
```

إن فهمت البرنامج السابق فنقول لك تهانينا .

الكلاسات الداخلية

الكلاس الداخلي او الكلاس المعشش هو الكلاس الذي يتم تعريفة ضمن مجال التغطية التابع لصنف اخر

```
class ammar{
        ammar(){System.out.println("Star ammar");}
        static class ammarlocal{
                        ammarlocal(){System.out.println("Star ammarlocal");}
                        }
        }
class aldopaee {
  public static void main(String[] args) {
    new ammar.ammarlocal();
                        }
    static{System.out.println("Star aldopaee");}
            }
```

```
D:\JCreator LE\GE2001.
Star aldopaee
Star ammarlocal
```

إذا تم إبعاد جملة static فماذا يكون خرج الكود ؟

- للطالب النبيل فقط ما خرج هذا الكود ولماذا

```
class ammar{
        ammar(){System.out.println("Star ammar");}
        static class ammarlocal{
```

```
                                        ammarlocal(){System.out.println("Star ammarlocal");}
                                        }
            ammarlocal b= new ammarlocal();
           }
  public class aldopaee {
   public static void main(String[] args) {
     new ammar.ammarlocal();
    }
     static{System.out.println("Star aldopaee");}
  }
```

- **وهل هذا الكود يشبه الكود السابق**

```
class ammar{
            ammar(){System.out.println("Star ammar");}
            static class ammarlocal{
                                        ammarlocal(){System.out.println("Star ammarlocal");}
                                        }
            ammarlocal b= new ammarlocal();
            {ammar b= new ammar();}
           }
public class aldopaee {
  public static void main(String[] args) {
    new ammar.ammarlocal();
  }
    static{System.out.println("Star aldopaee");}
}
```

- **انت معي بناتج هذا الكود**

```
D:\JCreator LE\GE2
Star ammar
show ammarlocal
```

```
  class ammar{
            ammar(){System.out.println("Star ammar");}
            static class ammarlocal{
                                        ammarlocal(){System.out.println("Star ammarlocal");}
                               static void show(){System.out.println("show ammarlocal");}
                                        }
            static{ammar b= new ammar();}
```

```
        }
public class aldopaee {
 public static void main(String[] args) {
  ammar.ammarlocal.show();
  }
}
```

- ان قلت ناتج الكود صحيح فقد أخطأت خطاءَ لن أسامحك به فاستطيع أن أقول انك لم تقرءا كتابي بتمعن وإنما تطلع على صفحاته وأنت مشغول البال فالله يعينك .

صلي على من بعت للصلاة علية وركز

صحيح إني قلت أن المترجم ينفذ الجمل static أولاً نعم ولكن في حالة إطلاق الهدف ففي كودنا السابق وصلنا وصول مباشر للكلاس الداخلي ولم ننشط داله البناء للكلاس الخارجي أفهمت ألان ويش الفائدة بعد ما أغضبتني !!!!!!!!!!!!!!!!!!!

- هذا الكود يبين أننا نستطيع استخدام متغيرات الكلاس الخارجي

```
class ammar{static int i=20;
        ammar(){System.out.println("Star ammar");}
        static class ammarlocal{static int i=40;
                                        ammarlocal(){int i=50;
                                          System.out.println(i+" "+this.i);
                                          System.out.println(ammarlocal.i+" "+ammar.i);
                                                }
                                        }
        }
public class aldopaee {
 public static void main(String[] args) {
  new ammar.ammarlocal();
  }
}
```

```
D:\JC
50 40
40 20
```

- هذا الكود لن ينفذ فاكتشف الخطاء واجعل خرجه نفس الكود السابق دون المساس بمحدد الوصول

```
class ammar{static int i=20;
         ammar(){System.out.println("Star ammar");}
```

```
D:\JC
50 40
40 20
```

```
        private static class ammarlocal{static int i=40;
                                          ammarlocal(){int i=50;
                                            System.out.println(i+" "+this.i);
                                            System.out.println(ammarlocal.i+" "+ammar.i);
                                                    }
                                          }
        }
public class aldopaee {
  public static void main(String[] args) {
   new ammar.ammarlocal();
   }
}
```

- **أيضاً للطالب النبيل لماذا هذا الكود لا ينفذ**

```
class ammar{int i=20;
         ammar(){new ammarlocal();}
         static class ammarlocal{static int i=40;
                                          ammarlocal(){int i=50;
                                            System.out.println(i+" "+this.i);
                                            System.out.println(ammarlocal.i+" "+ammar.i);
                                                    }
                                          }
        }
public class aldopaee {
  public static void main(String[] args) {
   new ammar();
   }
}
```

الكلمة (final) / بما معنى ثابت أي لا نستطيع تغيره ولها عدة استعمالات فان ذكرت مع

- **Final class / فإننا لا نستطيع توريث الكلاس .**
- **Final function / فإننا لا نستطيع عمل override .**

سيتم شرحهما لاحقاً

❖ Final variable / فإننا لا نستطيع تغير محتوى المتغيرات كهذا الكود خاطئ .

```
public class aldopaee {
  public static void main(String[] args) {
   ammar++;//error
   System.out.println(ammar);
   }
   static final int ammar=20;
}
```

حتى ألان المقدمة السابقة كنا نتحدث فقط عن الميزة الأولى للبرمجة كائنيه المنحى بقي الميزة الثانية والثالثة

الميزة الأولى كما قلت هي الكائنات وتصرفها كالواقع تماما

الميزة الثانية الوراثة

ما هو الوراثة Inheritance؟

يوجد في علوم الأحياء علم أسمه التصنيف العلمي (Scientific Classification) وفيه يقوم العلماء بتصنيف الكائنات الحية وترتيبها طبقاً للخواص المشتركة. أول نظام للتنصيف قام به أرسطو الذي صنفها على أساس بيئتها ، وقد ترجم ابن رشد تصنيف أرسطو في كتاب مفقود وبقيت الترجمة اللاتينية لكتاب ابن رشد. و التصنيف الحديث تعود جذوره إلى نظام كارلوس لينيوس، الذي صنف الأنواع طبقا للخواص الفيزيائية المشتركة.

يبدأ التصنيف الرئيسي بتسلسل مملكه ، شعبة ، طائفة ، رتبة ، عائلة ، جنس ، نوع . بعد ذلك أضيف فوق رتبة ، تحت رتبة ، فوق طائفة ، تحت طائفة ، قبيلة . وتصنيفات أخرى.

فمثلا الإنسان ينتمي لمملكة الحيوان (Animalia) ومن صفات هذه المملكة أن الكائنات فيها متعددة الخلايا multicultural أي إنها كائنات تتكون من أكثر من خلية و إذا تدرجنا في شجرة الحياة للإنسان فسنجده ينتمي إلى طائفة (class)أسمها الثديات (Mammilla) والتي ينتمي إليها كل الحيوانات و من صفات هذه

الطائفة أن الكائنات التي تنتمي إليها تلد صغاراً و ترضعهم أمهاتهم اللبن عن طريق الثدي. وينتمي الإنسان أيضاً إلى تحت طائفة (subclass) أسمها placentalia و التي من صفاتها أن الطفل في مرحلة الحمل يتغذى عن طريق المشيمة.

فالإنسان يرث صفات وسلوك ال تحت طائفة (subclass) التي تسمى placentalia وبالتالي يرث صفات وسلوك الطائفة (class) التي أسمها Mammaila وبالتالي يرث صفات وسلوك المملكة Animalia. وكما نري فإن التصنيفات العليا أي التي تتجه ناحية جذر الشجرة مثل المملكة تحتوي على صفات عامة و كلما تدرجنا ناحية فرع الشجرة كلما كانت الصفات و السلوك أكثر تخصصاً مثل صفات وسلوك الإنسان.

مفهوم ال Inheritance في OOP ينطبق عليه نفس الكلام السابق. فال class من الممكن أن ترث صفات و سلوك class أخرى. وال class التي ترث نسميها subclass و الclass التي يورث منها تسمى superclass وكما نلاحظ فهذه المسميات جاءت من علم التصنيف.

وفي الجافا من الممكن أن ترث ال class مباشرة من class واحدة فقط ولكنها تستطيع أن ترث من أكثر من class بطريقة غير مباشرة.

فلو قلنا أن الشجرة التي في الصورة السابقة هي شجرة كاملة (هي غير كاملة بالطبع) فإننا نقول أن ال subcalss التي تسمى placentalia ترث صفات ال class التي تسمى Mammaila بطريقة مباشرة و ترث صفات المملكة Animalia بطريقة غير مباشرة.

وكما نرى فإن ال subclass ليست محدودة بصفات ال superclass التي ترث منها بل تزيد عليها صفات وسلوك . في الجافا فإن ال superclass العليا في شجرة الوراثة أي الجذر هي ال class التي تسمى Object.

ال class Object ليس لها علاقة بمعنى object الذي تكلمنا عنه بل هي class وإسمها Object. وطالما أن جذر الشجرة في الجافا هي Object class فإن كل ال classes ترث سلوك وصفات هذه ال class مثل السلوك (method) toString و التي ترجع String (تتابع من الحروف) والتي تصف ال class.

لو رجعنا لعلم التصنيف و أردنا أن نعرف class تكون هي الجذر لكل الأشياء سواء حية أو غير حية فما هي ال methods التي ستكون في هذه ال class؟

أعتقد إنها تُسَبِّحُ.

يقول تعالى (تُسَبِّحُ لَهُ السَّمَاوَاتُ السَّبْعُ وَالأَرْضُ وَمَن فِيهِنَّ وَإِن مِّن شَيْءٍ إِلاَّ يُسَبِّحُ بِحَمْدَهِ وَلَـكِن لاَّ تَفْقَهُونَ تَسْبِيحَهُمْ إِنَّهُ كَانَ حَلِيماً غَفُوراً)

إن مميزات ال Inheritance هي:•

ال subclasses نستخدمها لنحصل على سلوك وصفات أكثر تخصصا من ال superclass التي سلوكها عاما

في البرامج الكبيرة يشترك أكثر من مبرمج في كتابة البرنامج ومن الممكن أن يكتب أحد المبرمجين abstract classes أي classes تجريدية بمعنى إنها تشتمل على سلوك عام وعلى المبرمجين الآخرين أن يقوموا في ال subclasses بكتابة الكود الخاص بهذه السلوك (methods) فمثلا في برنامج صناع الحياة قام المبرمج عمرو خالد بعمل ال abstract class التي أسمها نهضة و ال methods التي تنتمي لها هذه ال class هي (مثلا) : تنهض بالأب و تتفوق و تبدع و تصمم و تنتج

وعلى المبرمجين المشتركين في برنامج النهضة أن يكتبوا subclasses ترث من ال class نهضة و يقوموا بكتابة ال الكود الخاص بال methods التي تنتمي لل class نهضة. فمنهم من يكتب ال class زراعة الأسطح ومنهم من يكتب class النهضة الصحية وهكذا...

فالنعود لمثال الشخصية التي ذكرت في بداية شرح oop صفحة 32 لنقول أن الشركة المطورة للعبة طورت سابقا لعبة بها شخصيات أيضا لكن كتاب كائن الشخصية لم يطوروها بالشكل المطلوب فقط اكتفوا بجعلها تتصرف التصرفات الطبيعية المشتركة بين الشخصيات الطبيعة الحقيقية مثلا التنفس الحركة الطبيعة وحدود القطع في المشهد والتصادم

ثم "اتكنسل المشروع " وتم إغلاقه وإعلان فشله وبعد ٥ سنوات قام مشروع جديد وهو مشروع مبرمجنا الصنديد وتم تغيير أصناف المبرمجين في هذه الفترة لكن الوثائق القديمة والأدوات والفئات مازالت موجودة فكل ما سيفعله مدير المشروع هو أن يهرع بجلب الفئة التي عرفت سابق ويضيفها للمشروع بحال كائن أب ويشتق فريق تطوير شخصية المحارب منها ليكتسب كل الصفات الأساسية في لمح البصر ويصبو مجهودهم على الإضافات فقط كالقتال وغيره !!!

يمكن لفرق الشخصيات إنتاج مئات الشخصيات بسرعة خرافية لأنهم سيركزون على الإضافات فقط مثلا شخصية طباخ وشخصية لاعب كرة وشخصية وحش "إذا كان يتنفس أيضا !! " كل هذه الشخصيات ستتطور بسرعة كبيرة

بالطبع مبرمجنا سيكون سعيدا للغاية وهو يحتسي قدحا من القهوة وهو يراقب هذه الفرق وهي تعمل فهو بلا عمل لأنه أنجز كل عمله !!!

فإعادة الاستخدام ميزة رائعة جدا لكن تتطلب بعض الحذر واتساع الأفق وبعد النظر

ومحاولة جعل التطور يكون في أكبر عدد من المستويات لكل مستوى فئة ترث من التي قبلها ليسهل في أي لحظة الوصول للفئة الأقرب للحاجة .

الكلمة ألمفتاحيه (extends) / وهي أساس الوراثة فعند وضع هذه الكلمة بجانب اسم الكلاس هذا يعني أن هذا الكلاس يرث من كلاس أخر

```
class aldopaee {static int i=10;
                static private int j=20;
                static protected int c=100;
               static int r(){return j;}
              }
class ammar extends aldopaee {
 public static void main(String[] args) {
  System.out.println(i);
  System.out.println(c);
  System.out.println(r());
  }
}
```

هنا عملنا عملية تورث للكلاس ammar من الكلاس aldopaee
فأصبح الكلاس الوارث ammar يستطيع أن يصل لجميع الدوال والمتغيرات التي من نوع public, protected في الكلاس aldopaee

الكلمة ألمفتاحيه (Super) / ذكرنا سابقاً أن الكلمة this تشير إلى الكلاس نفسه فالكلمة super تشير إلى نفس صنف الأب واقصد بالأب أي الكلاس المورث أي موقعة في مثالنا السابق aldopaee ولها استخدامان

i) تستدعي باني الصنف الأب .

```
class ammar{
              ammar(){System.out.println("Star ammar");}
```

```
        }
  class aldopaee extends ammar{
   public static void main(String[] args) {
    new aldopaee(10);
    System.out.println("End");
    }
    aldopaee(int i){ super(); System.out.println(i*i);}
  }
```

```
Star ammar
100
End
```

ويجب مرعاه الآتي

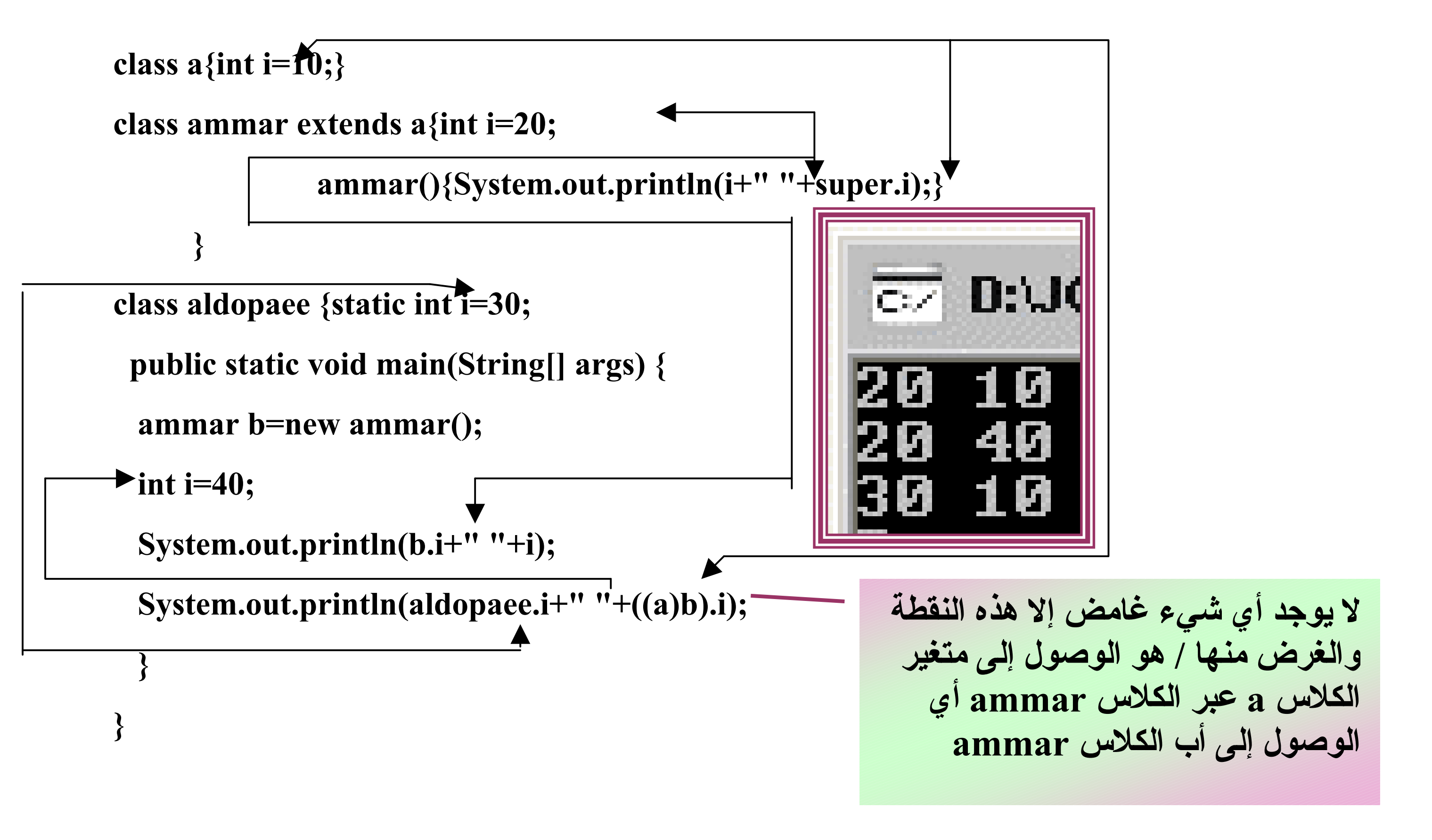
- عند استدعاء دالة بناء كلاس الأب فيجب وضع الكلمة super في بداية دالة البناء .

(١) تستدعي دوال ومتغيرات الأب .

```
class a{protected int i=10;}
class ammar extends a{protected int i=20;
                      ammar(){}
                      void test(int a){
                                          System.out.println(super.i);
                                          System.out.println(this.i);
                                          System.out.println(i);
                                         } }
class aldopaee {static int i=30;
 public static void main(String[] args) {
  ammar b=new ammar();  b.test(40);
 }}
```

```
10
20
20
```

• وهذا كود أخر



• أسبقية استدعاء دوال البناء للكلاس الموروث

عند تنشيط كلاس وهو يرث من كلاس أخر فان أولا يتم تنفيذ دالة بناء الكلاس الأب ثم دالة بناء الكلاس المنشط وهذا الكود يبين ذلك

```
class a{
        a(){System.out.println("star a");}
        }
class b extends a{
                b(){System.out.println("star b");}
                }
class ammar extends b{
```

```
D:\JCreator
star a
star b
star ammar
```

```
                    ammar(){System.out.println("star ammar");}
                    }
    class aldopaee {
     public static void main(String[] args) {
      ammar b=new ammar();
     }
    }
```

- أم إذا تواجدت دوال تنشيط أي إنشاء هدف فيختلف الأمر ،

```
class ammar {
   ammar() {System.out.println("ammar");}
        }
class ammar2{
   ammar2() {System.out.println("ammar2");}
        }
class aldopaee extends ammar2 {
 ammar a=new ammar();
 public static void main(String[] args) {
  new aldopaee();
 }
 aldopaee() {System.out.println("aaldopaee");}
}
```

١- يتم إطلاق الكلاس الرئيسي .
٢- يتم الانتقال لكلاس الأب ammar2 وينفذ دالة البناء الخاصة به .
٣- يتم تنفيذ جمل المتغيرات لتنشيط الكلاس
٤- ينفذ داله البناء للكلاس الرئيسي aldopaee .

```
D:\JCreato
ammar2
ammar
aaldopaee
```

نلاحظ في كودنا السابق لم نذكر الكلمة ألمفتاحيه super وكما قلنا أنها تعمل على استدعاء دوال البناء فعندما تذكر هذه الكلمة فإنها تستدعي دالة البناء الخاصة بنفس نوع super .

```
    class a{
            a(){System.out.println("star a");}
            a(int i){System.out.println("a ="+i*i);}
           }
    class b extends a{
                    b(){super(10);System.out.println("star b");}
                   }
    class ammar extends b{
```

```
D:\JCreator
a =100
star b
star ammar
```

```
                              ammar(){System.out.println("star ammar");}
                              }
class aldopaee {
  public static void main(String[] args) {
   ammar b=new ammar();
  }
}
```

- قاعدة / في حال تنشيط الكلاس وهذا الكلاس يرث من كلاس أخر وفي داخل الكلاس دوال تنشيط أي كتل بناء استاتيكية فان أسبقية التنفيذ لكتل البناء الاستاتيكية ثم دوال البناء وهذا الكود يبين ذلك

```
class b {
        b(){System.out.println("star b");}
        b(int i){System.out.println("b ="+i*i);}
      }
class ammar extends b{
                              ammar(){System.out.println("star ammar");}
                              static{b m= new b(10);}
                              }
class aldopaee {
 public static void main(String[] args) {
  ammar b=new ammar();
 }
}
```

```
D:\JCreator L
b =100
star b
star ammar
```

- وراثة الكلاس الداخلي

```
class a {
      a(){System.out.println("a");}
      class si{
              si(){System.out.println("si");}
              }
      }
class ammar extends a.si{
        ammar(){new a().super();System.out.println("star Ammar");}
```

```
D:\JCreator
a
si
star Ammar
End main
```

```
                }
class aldopaee{
public static void main(String args[]) {
ammar d=new ammar();
System.out.println("End main");
}}
```

- على الرغم من إننا لم نحفز دالة البناء للكلاس a ولكن نجد من أنها تم تنفيذها فالسؤال هنا إذا تم إبعاد دالة البناء للكلاس a فهل سينفذ البرنامج وما خرج البرنامج ولماذا ؟
- في هذا الكود أولوية بالتنفيذ فتتبعه بتمعن

```
class a {
        a(){System.out.println("a");}                2
        a(int g){System.out.println("a ="+g);}
        class si{
                si(){System.out.println("si");}          5
                a s=new a(10);                 4
                }
        }
class ammar extends a{
                ammar(){System.out.println("star Ammar");}      6
                si d=new si();                 3
                static {System.out.println("star ammar");}     1
                }
class aldopaee{
public static void main(String args[]) {
ammar d=new ammar();          0
System.out.println("End main");      7
}
}
```

D:\JCreator LE\(

```
star ammar
a
a =10
si
star Ammar
End main
```

- تجاهل دوال كلاس الأب وعمل عملية method overriding

والقصد بة وجود دالة موجودة بالكلاس الأب وموجودة بالكلاس الابن بنفس الاسم ففي هذه الحالة يحصل عملية تعديل فعند استدعاء هذه الدالة فيأخذ المترجم دالة الابن بدل دالة الأب و هذا الكود يبين ذلك

```
class a {
        a(){}
       void show(){System.out.println("halo a");}
      }
class ammar extends a{
                ammar(){}
                void show(){System.out.println("halo ammar");}
               }
class aldopaee{
public static void main(String args[]) {
ammar d=new ammar();
d.show();
System.out.println("End main");
}
}
```

```
D:\JCreator LE
halo ammar
End main
```

- نجد في كودنا السابق انه تجاهل الدالة الموجودة في الكلاس الأب a طيب السؤال هنا كيف يمكنني استخدام دالة الكلاس الأب أي بمعنى كيف يمكنني أن أصل إلى ألداله الموجودة في الكلاس الأب ؟ للجواب على هذا السؤال هو أن تصلي على نبيك وتتبع هذا الكود

```
class a {
        a(){}
      static void show(){System.out.println("halo a");}
      }
class ammar extends a{
                ammar(){}
               static void show(){System.out.println("halo ammar");}
               }
class aldopaee{
public static void main(String args[]) {
ammar d=new ammar();
((a)d).show();
```

```
D:\JCreat
halo a
End main
```

هنا منطقة التحكم بعد ما جعلنا الدوال من نوع static فلو أردنا الوصول لدالة كلاس الابن نكتب d.show()

```
System.out.println("End main");
}
}
```

• وهذا الكود يوضح أكثر الاختلاف بين عمليه التحوير أي التعطيل في الدوال static والعادية

```
class a {
    a(){}
   static String show1(){return "Ammar";}
   String show2(){return "Mohammed";}
  }
class ammar extends a{
        ammar(){}
         static String show1(){return "Sand";}
         String show2(){return "Alopaee";}
       }
class aldopaee{
public static void main(String args[]) {
a d=new ammar();
System.out.println(d.show1()+"  "+d.show2());
}
}
```

D:\JCreator LE\G
Ammar Alopaee

في هذا التعريف لا يتم استبدال الدوال التي تحمل نفس الاسم بل يذهب إلى دالة الأب وينفذها ويترك حق الابن أي لا توجد overload

• توريث كلاس و التعديل فيه

والمعنى هو اخذ جميع صفات كلاس الأب وإضافة صفات نحتاجها فوق صفات كلاس الأب وهذا الكود يبين ذلك .

```
class a {int i;
        a(int i){this.i=i;}
        int get_i(){return i;}
       }
class ammar extends a{int a,b;
          ammar(int a,int b,int c){super(c);this.a=a;this.b=b;}
          void show(){System.out.println(super.get_i()+" "+a+" "+b);}
            }
class aldopaee{
```

D:\JCreato
30 10 20

```
public static void main(String args[]) {
ammar d=new ammar(10,20,30);
d.show();
}
}
```

- ألان بعد ما تعلمنا جميع أساسيات الوراثة وأساسيات oop فينبغي عليك إيجاد خرج هذا الكود من تلقاء نفسك وان لم تعرف فنصيحتي لك ان ترجع قراءة الكتاب من البداية .

```
final class a {int i;
         a(int i){this.i=i;}
         int get_i(){return i;}
        }
class ammar extends a{int a,b;
                      ammar(int a,int b,int c){super(c);this.a=a;this.b=b;}
                      void show(){System.out.println(super.get_i()+" "+a+" "+b);}
                     }
class aldopaee{
public static void main(String args[]) {
ammar d=new ammar(10,20,30);
d.show();
}
}
```

- هل هذا الكود خرجه كذا

```
class a {
        a(){System.out.println("star a");}
        a(int x){System.out.println("star a="+x);}
        void get(int x){System.out.println("get a="+x);}
    }
class ammar extends a{static a b1,b2;
         ammar(){System.out.println("star ammar");}
         static{b1=new a(1);b2=new a(2);}
         }
class aldopaee{
public static void main(String args[]) {
ammar d=new ammar();
System.out.println("star aldopaee");
```

```
D:\JCreator LE\G
star a=1
star a=2
star a
star ammar
star aldopaee
star a
star ammar
get a=10
```

تعتبر هذه الحالة
كلاس داخل كلاس

```
ammar.b1.get(10);
}
}
```

- إن قلت صح فقد أخطئت بسبب أننا هنا وصلنا وصول مباشر لذلك تم تجاهل دالة البناء الأب والابن أي لا توجد تنشيط لهما .
- ملاحظة محددات الوصل التي شرحتها سابقاً يرجى الانتباه منها في الوراثة كهذا الكود يوجد بة خطاء قم بإيجاده .

```
class A {
    int i;
    private int j;
    void setij(int x, int y) {i = x;j = y;}
   }
class B extends A {
int total;
void sum() {total = i + j; }
          }
class aldopaee {
public static void main(String args[]) {
B subOb = new B();
subOb.setij(10, 12);
subOb.sum();
System.out.println("Total is " + subOb.total);
}
}
```

- قبل أن ننتقل إلى الميزة الثالثة في الوراثة أهديكم هذا الكود الرهيب فمن فهمة أقول أنة قد فهم ما كتبته كاملاً .

```
class B {
     B(){System.out.println("B 6");}
      {System.out.println("B 5");}
      static {System.out.println("B 3");}
   }
```

```
D:\JCre
ammar 1
ammar 2
B 3
A 4
B 5
B 6
A 7
A 8
```

```
class A extends B {
        A(){System.out.println("A 8");}
        {System.out.println("A 7");}
        static {System.out.println("A 4");}
    }
class aldopaee {
public static void main(String args[]) {
new aldopaee();
}
  aldopaee(){new A();}
  {System.out.println("ammar 2");}
  static {System.out.println("ammar 1");}
}
```

١- ينفذ الكتل الاستاتيكية في البرنامج الرئيسي
٢- ينفذ الكتل الغير استاتيكية .
٣- ينفذ الكتل الاستاتيكية للكلاس الأب .
٤- ينفذ الكتل الاستاتيكية للكلاس الابن .
٥- ينفذ الكتل الغير استاتيكية للكلاس الأب .
٦- ثم دالة البناء للكلاس الأب .
٧- ينفذ الكتل الغير استاتيكية للكلاس الابن .
٨- ثم دالة البناء للكلاس الابن .

- إن فهمت الكود السابق ينبغي عليك أن توجد خرج هذا الكود دون استعمال المترجم فهذا الكود شبيه للسابق .

```
class B {
         B(){System.out.println("B 6");}
         {System.out.println("B 5");}
         static {System.out.println("B 3");}

        }
class A extends B {
                     A(){System.out.println("A 8");}
                     {System.out.println("A 7");}
                     static {System.out.println("A 4");}
                    }
class aldopaee {
public static void main(String args[]) {
new aldopaee();
}
    aldopaee(){new A();}
    {System.out.println("ammar 2");}
    static {System.out.println("ammar 1");}
    A b=new A();
    {B a=new B();}
}
```

إرسال الطرق ديناميكياً Dynamic method dispatch

ويقصد بة إرسال الطرق ديناميكياً في حالة overriding أي التحكم في الوصول للدوال

```
class A {
        void callme() {System.out.println("Inside A's callme method");}
       }
class B extends A {
                   void callme() {System.out.println("Inside B's callme method");}
                 }
class C extends A {
                 void callme() {System.out.println("Inside C's callme method");}
                 }
class aldopaee {
public static void main(String args[]) {
A a = new A();
B b = new B();
C c = new C();
A r;
r = a;
r.callme();
r = b;
r.callme();
r = c;
r.callme();
}
}
```

```
D:\JCreator LE\GE2001.exe
Inside A's callme method
Inside B's callme method
Inside C's callme method
```

الفئات المجردة Abstract

والمعنى منة الفئات الخالية من الكود أي تحتوي على توقيع فقط أي يبقى بدون تنفيذ أي نستطيع أن نقول النوع المجرد من البيانات ليس له وجود في الواقع إنما هو في الحقيقة مفهوم أو فكرة للأصناف الأخرى التي تتشابه فمثل هذا الكود نلاحظ ان الكلاس B من نوع مجرد والدالة show من نوع مجرد فجميع الكلاسات التي ترث كلاس B تعيد تعريف الدالة show كما تشاء ففي الكلاس A جعلناها تطبع نسبة الربح مثلاً 10. وفي الكلاس C جعلناها تطبع نسبة الربح 15ً. فهنا يتبين فائدة الأصناف المجردة أي تتيح للعملاء وضع الكود الخاص بهم .

```
abstract class B {int i;
        B(int s){i=s;}
        abstract void show();
       }
class A extends B {
                        A(int s){super(s);}
                        void show(){System.out.println(super.i*.10);}
          }
class C extends B {
                        C(int s){super(s);}
                        void show(){System.out.println(super.i*.15);}
          }
class aldopaee {
public static void main(String args[]) {
A a=new A(100);
C b=new C(100);
a.show();
```

```
b.show();
}
}
```

قواعد حول التجريد

- لا يمكن وضع طريقة مجردة بداخل كلاس غير مجرد كهذا الكود .

```
class B {int i;
        B(int s){i=s;}
        abstract void show();
       }
```

- يجب على الكلاس الغير مجرد الذي يرث من كلاس مجرد تنفيذ جميع الطرق حتى ولو لم يستخدمها كهذا الكود لن ينفذ إطلاقاً.

```
abstract class B {int i;
        B(int s){i=s;}
        abstract void show();
        abstract void show2();
       }
class A extends B {
                        A(int s){super(s);}
                        void show(){System.out.println(super.i*.10);}
          }
```

- لا يمكن الحصول على فئات مشتقة من الكلاس المجرد باستخدام الكلمة **new** كهذا الكود خاطئ .

```
abstract class B {int i;
       B(int s){i=s;}
       abstract void show();
      }
class aldopaee {
public static void main(String args[]) {
B a=new B(100);
}
}
```

- من الممكن التصريح على كلاس مجرد يحتوي على طرق غير مجردة .

```
class B {
        B(){System.out.println("B");}
        void show(){}
       }
abstract class A extends B {int i;
                                        A(int s){super();}
                                        abstract void show();
                          }
class C extends A {
                    C(int s){super(s);}
                    void show(){System.out.println(super.i*.15);}
                   }
class aldopaee {
public static void main(String args[]) {
C a=new C(100);
}
}
```

- من الممكن إن يكون الكلاس الابن مجرداً حتى ولو كان الكلاس الأب غير مجرد .

```
class B {int i;
        B(int s){i=s;}
        void show(){System.out.println(super.i*.15);}
       }
abstract class A extends B {
                                        A(int s){super(s);}
                                        void show(){System.out.println(super.i*.10);}
            }
```

والميزة الثالثة

وتعدد الأشكال polymorphism

يسمح لنا بكتابة برنامجنا في صورة قابلة لتغيير واسع النطاق؛ سواء كان التغيير لفئات موجودة مسبقاً أو تغيير مستقبلي لإنتاج برامج جديدة. هذه الخاصية تسهل علينا توسيع قدرات نظامنا .

وكما ذكرنا في الأعلى أن الفئات الجديدة ـتسمى فئة فرعية -*subclass* ترث صفات الفئات التي أنتجت وتكونت منها ـتسمى الفئة الأب -*superclass* كما يرث الطفل جينات أبويه. وهذه الفئة الجديدة والتي تعتبر *subclass*، من الممكن أن تكون *superclass* لفئات جديدة أخرى ينشئها المبرمج. وهكذا تمتد لدينا سلسلة من الوراثة بين الفئاتextends ، يحكمها قانون" الوراثة المفردة "Single Inheritance حيث ينص هذا القانون على:
تنشأ أي فئة فرعية من فئة أم واحدة، فالجافا لا تدعم التوارث المتعدد multiple inheritance كالسي++ ولكنها تدعم مفهوم الواجهات Interfaces ، التي حان الوقت لشرحها ، فنظام الواجهات يساعد الجافا على تحقيق فائدة التوارث المتعدد مع عدم وجود الأخطاء المترابطة الناتجة عن هذا التوارث المتعدد !

تذكر أن أي كائن ينتمي إلى فئة فرعية فهو ينتمي إلى الفئة الأب لهذه الفئة الفرعية ويحمل خصائصهما وسلوكهما.

الواجهات Interfaces

هي عبارة عن بيئة مشابهة للكلاس وتحتوي فقط على الطرق المجردة وبمعنى أخر يمكن القول أن الواجهات مشابهة للكلاسات المجردة

الهيكل العام للواجهات

اسم الواجهة interface معدل الوصول

وعند وراثه الكلاس من الواجهه نستخدم الكلمة implements

```
interface a{int i=100;
            int get();
        }
class ammar implements a {
                        ammar(){}
                        public int get(){return i;}
              }
class aldopaee {
public static void main(String args[]) {
        ammar t=new ammar();
```

```
System.out.println(t.get());
}
}
```

قواعد حول الواجهات

- **من الممكن أن تحتوي الواجهة على متغيرات ودوال مثله مثل الكلاس ولكن كل المتغيرات فيه تكون نهائية أي final لا يمكن تغير قيمتها ، وجميع الطرق من نوع abstract حتى ولم تذكر .**
- **تتألف جميع الطرق في الواجهات من التوقيع فقط ولا تحتوي على التنفيذ .**
- **لابد من إعادة تعريف جميع الدوال المعرفة بداخل الواجهة في الكلاس وإلا اعتبر هذا الكلاس من نوع مجرد فلا يمكن اشتقاق هدف منة ٠**
- **تسمح لغة الجافا بالوراثة المتعددة بواسطة الواجهات .**

```
interface a{int i=100;
                    int get();
                }
interface b{int j=10000;
                    int Get();
                }
class ammar implements a,b {
                                        ammar(){}
                          public int get(){return i;}
                          public int Get(){return j;}
                 }
class aldopaee {
public static void main(String args[]) {
       ammar t=new ammar();
System.out.println(t.get()+" "+t.Get());
}
```

```
}
```

- تستطيع الواجهة وراثة واجهه أو أكثر .

```
interface a{int i=100;
                        int get();
                 }
interface b extends a{int j=10000;
                                int Get();
                         }
class ammar implements b {
                                                  ammar(){}
                              public int get(){return i;}
                              public int Get(){return j;}
                  }
class aldopaee {
public static void main(String args[]) {
        ammar t=new ammar();
System.out.println(t.get()+" "+t.Get());
}
}
```

وبعد هذه المقدمة وهذا التوصيف لعالم الـ OOP نلاحظ أن كلّ التركيز في هذا النوع من البرمجة يقع على الـفئات Classes ، فالمبرمج يستخدم الفئات المبنية مسبقاً في اللغة مع الفئات التي يبنيها هو كي ينتج برنامجاً بالجافا، ربما يفسر هذا الاسم OOP .

الحزم Packages

حزم الجافا (Java Packages)

- **ما هي حزم الجافا؟**
- **لماذا نحتاج حزم الجافا؟**
- **كيف نستطيع إنشاء حزم الجافا؟**

ما هي حزم الجافا؟

التعريف / حزم الجافا هي مجموعة من الفئات المترابطة، و كل مجموعة من الفئات تنظم تحت حزمة معينة لأجل تحديد الهوية . و الحزمة تتكون من:

- **حزم فرعية تحت الحزمة الأب .**
- **مجموعة من الفئات المتعلقة بالحزمة الأب .**

بعض الأمثلة: الحزمة Java تحتوي على حزم فرعية منها applet io alng net awt & util و لو أخذنا الحزمة الفرعية Java.awt لحصلنا على حزمة فرعية من awt مثل image و يكون الأبتداد لها Java.awt.image

لماذا نحتاج حزم الجافا؟

مبرمجي الجافا يعتمدون على الحزم لتكوين فئات مترابطة داخل هذه الحزم و الأسباب هي:

- **العثور على الفئات بشكل سريع و استخدامها بالبرامج .**
- **تنحدر الفئات تحت الحزم لكي لا تتعارض أسماء الفئات مع بعضها البعض .**
- **للتحكم بالفئات بشكل كامل .**

مسميات الحزم و الحزم الفرعية و الفئات/ الحزمة تتكون من حزم فرعية و فئات متفرعة، لكن لا نستطيع تسمية الحزمة أو الحزم الفرعية أو إحدى الفئات باسم واحد. و مثال على ذلك: الحزمة java.awt لديها حزمة فرعية بالاسم image. لكن لا نستطيع تسمية إحدى الفئات بالاسم image ، لأن الاسم محجوز للحزمة الفرعية و العكس صحيح.

كيف نستطيع إنشاء حزم الجافا؟

لنفترض ألان إننا سننشئ كلاس به دالة اسمها show(); مهمتها طباعة رسالة على الشاشة ونريد استخدام هذه الدالة في كلاس أخر .

أولاً ننشئ الكلاس وبداخلة دالة الطباعة وفي بداية الكلاس نكتب (اسم الحزمةpackage)

```
package myammar;

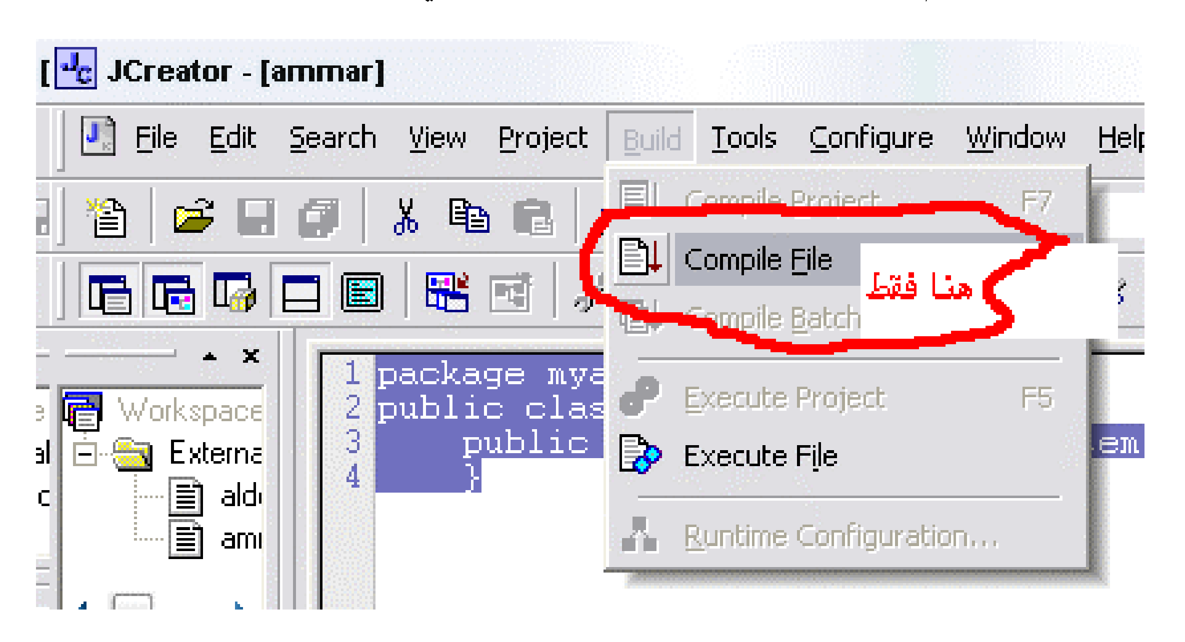
public class ammar{

        public static void show(){System.out.println("my ammar");}

        }
```

نعمل عملية ترجمة وهنا تتم الترجمة على حسب المترجم المستخدم

فان كنت تستخدم JCreator فاعمل هكذا فقط كما في الصورة



وان كنت على dos اكتب الأتي javac –d . ammar.java

لنرى لو لدينا مجموعة من الفئات و التي نستطيع وضعها في حزمة معين ة. نفترض إننا كتبنا فئات عن النقاط و الدائرة و المستطيل و المربع.

الآن نود أن نضع هذه الفئات مع بعضها البعض في حزمة لعدة أسباب:

- **نستطيع نحن و المبرمجين الآخرين أن نجد هذه الفئات لأنها مترابطة .**
- **نستطيع نحن و المبرمجين الآخرين أن نعرف كيف نجد هذه الفئات لأنها دوال رسم مترابطة .**

- أسماء الفئات السابقة لن تتعارض مع أسماء الفئات من الحزم الأخرى لأنها سوف تكون تحت حزمة جديدة من إنشائك، مثال على ذلك :

<table>
<tr>
<td>

```
package geometry;

public class Rectangle extends
Point
{
   double width;
   double height;

   public Rectangle(int x int y
double w double h)
   {
      super(x y);
      width = w;
      height = h;
   }
}
```

</td>
<td>

```
package geometry;

public class Point {
    int_x coord;
    int_y coord;

   public Point() {
      x_coord = 0;
      y_coord = 0;
   }
   public Point(int x int y) {
      x_coord = x;
      y_coord = y;
   }
}
```

</td>
</tr>
<tr>
<td>

```
package geometry;

public class Square extends
Point
{
   double edge;

   public Square(int x int y
double e)
   {
      edge = e;
   }
}
```

</td>
<td>

```
package geometry;

public class Circle extends Point
{
   double radius;

   public Circle(int x int y double r)
   {
      super(x y);
      radius = r;
   }
}
```

</td>
</tr>
</table>

نلاحظ هنا أننا أضفنا السطر package geometry في كل الفئات) كل فئة توجد في ملف مستقل). لكن لو فرضنا أننا نريد استخدام الفئة Rectangle موجودة بالحزمة java.awt مع الفئة الموجود بالحزمة geometryبنفس البرنامج الذي نريد كتابته، فماذا نفعل ؟

استدعاء فئتين بنفس المسمى / نستطيع ذلك باستخدام fully qualified name و هو كتابة المسار الكامل للفئة، مثال على ذلك:

```
java.awt.Rectangle rec1 = new
```

```
java.awt.Rectangle(...); // استخدمنا المسار الكامل للفئة

geometry.Rectangle rec2 = new
geometry.Rectangle(...); // هنا ايضاً و
```

كيفية استدعاء فئة معينة من الحزمة الخاصة بها / تستطيع استدعاء الفئات من الحزم عن طريق ثلاث طرق:

- **استدعائها عن طريق كتابة المسار الكامل (كما المثال السابق).**
- **استدعائها فقط عن طريق الحزمة java.awt.Rectangle**
- **استدعاء الحزمة كاملة بما فيها من فئات أخرى .;*.java.awt**

النجمة (*) تدل على استدعاء الحزم الفرعية و الفئات الموجودة تحت هذه الحزمة.

- **ملاحظة / عندما تنشئ حزمة وتريد الوراثة منها أي بالذي بداخلها فيجب عليك جعل اسم الكلاس والدوال التي ستستخدمها ودوال البناء من نوع public , protected .**

معالجة الاستثناءات

Exception Handling

مقدمة :

الاستثناء هو مؤشر لحدوث خطأ أثناء عملية تنفيذ البرنامج مما يؤدي إلى تعطيل التسلسل الطبيعي لتعليمات البرنامج وقد تعلمنا في الفصل السابق أن الوراثة في لغة الجافا تعطيها صفة الامتدادية وهذه الصفة يمكن أن تزيد من عدد ونوع الأخطاء التي يمكن أن تحدث حيث إن كل فصيلة جديدة تضاف إلى البرنامج يمكن أن تضيف مصدراً من مصادر الاستثناءات في البرنامج. إذاً نستطيع القول أن الاستثناء هو حدوث خطأ ما وهذا الخطأ ليس خطأ في بناء الجملة syntax error ولكنه قد يكون له العديد من المصادر مثل القسمة على صفر ومعاملات غير متاحة للدالة و الإشارة إلى عنصر في المصفوفة خارج نطاقها.

عند حدوث استثناء يحتاج البرنامج إلى معالجة هذا الاستثناء لكي يستمر تنفيذ البرنامج بصورة طبيعية وسابقا قبل عام ١٩٩٠ كانت معالجة الاستثناءات تتم باختبار قيم صحيحة تعود بدلائل مثل القيمة صفر تدل على النجاح والقيمة السالبة تدل على نوع من الاستثناءات وهذه القيم أصبحت تعرف بشفرات الأخطاء Error codes وقد تم اكتشاف أن استخدام هذا النوع من معالجة الأخطاء يتسبب في ثلاث مشاكل:

١ - غالباً تهمل شفرة الخطأ

٢ - اختبار شفرة الأخطاء تعترض التدفق الطبيعي للبرنامج مما يصعب تتبع المستخدم للبرنامج

٣- اختبار شفرة الأخطاء يزيد من حجم البرنامج

أساسيات معالجة الاستثناء في لغة الجافا

The basics of java Exception Handling

لقد أدت مشاكل استخدام شفرة الأخطاء Error codes إلى تطوير آلية جديدة لمعالجة الاستثناءات في لغة الجافا تعتمد على الكائنات مما أدى إلى برامج سهلة القراءة والتتبع وكذلك برامج أكثر مرونة.

وفي هذا النموذج عند حدوث استثناء أثناء تشغيل برنامج الجافا إما البرنامج program أو آلة لغة الجافا الافتراضية JVM تنشىء كائن لوصف الاستثناء ويشمل هذا الكائن قيم المتغيرات في لحظة حدوث الاستثناء.

إذا تم إنشاء الكائن من البرنامج فإن البرنامج يمرر ذلك الكائن إلى آلة الجافا الافتراضية JVM وعند استقبال الكائن تبحث في البرنامج عن معالج الاستثناء handler exception الذى يمكن أن يعالج الاستثناء الموصوف بالكائن. إذا وجد المعالج يتم تمرير الكائن لمعالج الاستثناء الذي يقوم باستخدام محتويات الكائن لمعالجة الاستثناء. إذا لم يوجد معالج الاستثناء يتوقف البرنامج عن التنفيذ.

لنأخذ هذا المثال البسيط .

```
String str = "x";
int i = Integer.parseInt(str);
System.out.println(str);
```

في السطر الأول أنشأت متغير من نوع نصي ووضعت فيه القيمة x

في السطر الثاني أنشأت متغير من نوع رقم صحيح وقرأت القيمة الرقمية من المتغير النصي. طبعاً في الحالات العادية من المفترض أن تكون القيمة الموجودة في النص رقم ..مثلاً "١٢٣" ولكن في حالتنا كانت حرف وليس رقماً.. لذا عملية القراءة ستتسبب حدوث exception كما يلي :

```
Exception in thread "main" java.lang.NumberFormatException: x
        at java.lang.Integer.parseInt(Integer.java:405)
        at java.lang.Integer.parseInt(Integer.java:454)
        at TryException.main(TryException.java:9)
Press any key to continue . . .
```

ما حدث في هذه الحالة نسميه .Exception وما ترونه في الصورة الأخيرة هي وسيلة الآلة التخيلية لإخبارنا أنها واجهت مشكلة، ولا تعرف كيف يمكن أن تحلها. وفي الحقيقة في الصورة ستجدون اسم الـ Exceptionوهو .NumberFormatException :

وفي الواقع مثالنا هذا

```
public class aldopaee{

        public static void main(String args[]){

        int i=0;

        i=i/i;

                }

        }
```

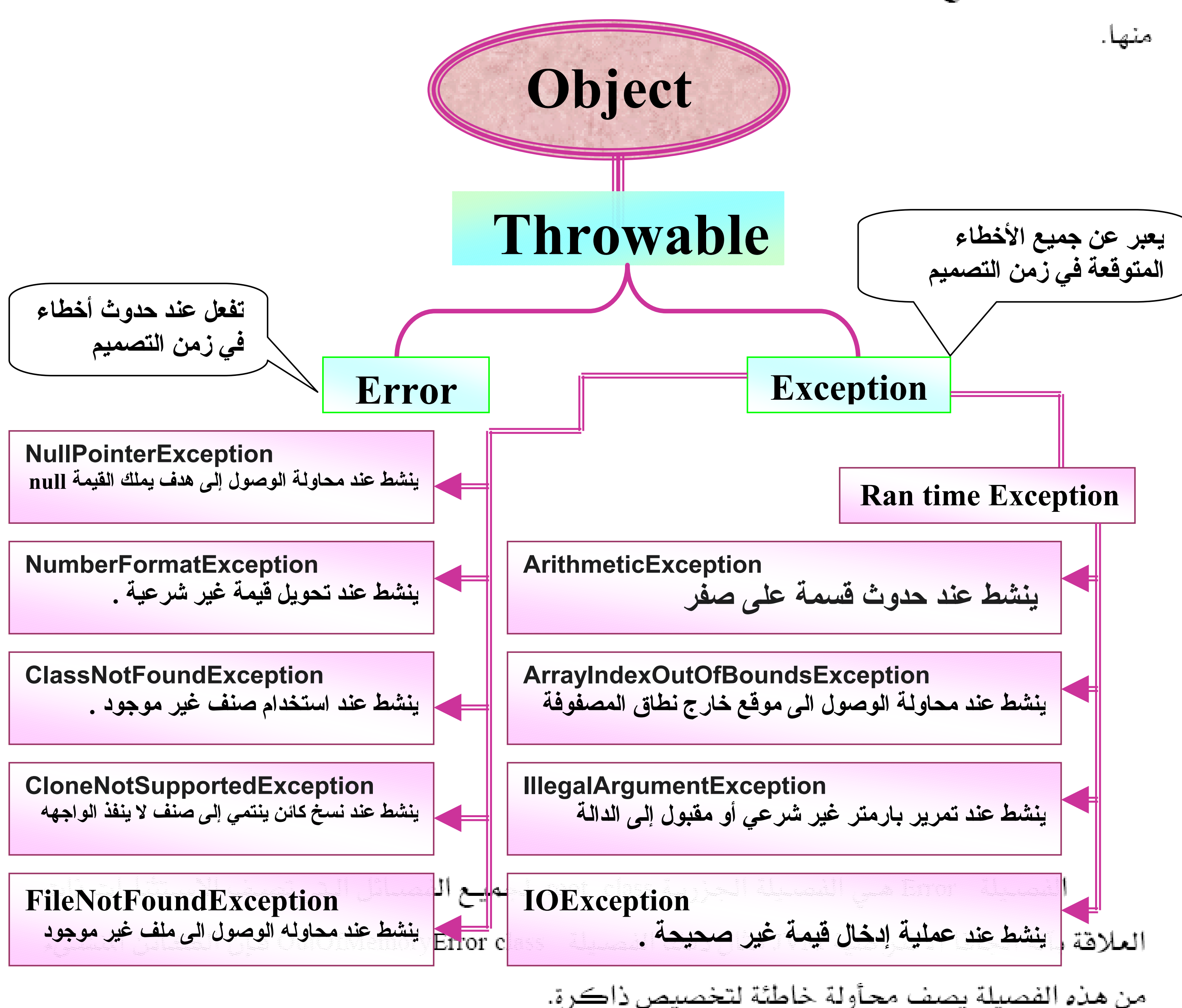
```
D:\JCreator LE\GE2001.exe
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at aldopaee.main(aldopaee.java:4)
```

نلاحظ أن تم إطلاق خطاء وهو القسمة على صفر فأنت في هذه الحالة قد تعالج هذا الخطاء بالمعالجة المناسبة كطباعة رسالة للمستخدم بأنه حصل خطاء وتعاود إدخال القيم مرة أخرى دون توقف البرنامج ففي مثالنا السابق يتم إيقاف البرنامج كلياً فلهذا وجدت الاستثناءات .

أنواع الاستثناءات Exception types

من الأمثلة السابقة تبين أن هناك العديد من أنواع الاستثناءات ومن معرفتنا للغة الجافا بأنها تتكون من فصائل فإن الاستثناءات في الجافا هي فصائل classes وكل فصيلة class تختص بنوع من الاستثناءات وجميع هذه الفصائل ترث الفصيلة العليا Throwable وتوجد فصيلتان فرعيتان ترثان هذه الفصيلة وهما subclass Exception و subclass Error وهذه الفصائل موجودة في الحزمة java.lang وهذه الفصائل الفرعية تصنف الاستثناءات أهي ذات علاقة بالبرنامج program related أم هي ذات علاقة بآلة الجافا الافتراضية JVM

الفصيلة Exception هي الفصيلة الجزرية root class لجميع الفصائل التي تصف جميع الاستثناءات ذات العلاقة ببرنامج جافا ويبين الجدول (٢ -١) بعض الفصائل الفرعية للفصيلة Exception ووصف كل منها.



العلاقة

من هذه الفصيلة يصف محاولة خاطئة لتخصيص ذاكرة.

خطوات إنشاء الاستثناءات

١- تعريف الاستثناء أو التصريح عنة   void bmw()throws نوع الاسثناء

٢- دفع الاستثناء   throw new ----------

٣- جلب الاستثناء ومعالجته   catch(-------)

إيعازات الاستثناء

١- try / يستخدم لتحديد المنطقة التي ستقع فيها الخطاء في البرنامج .
٢- Catch / تأتي بعد التعليمية السابقة مباشرة لتحدد نوع الخطاء المتوقع وقد يأتي أكثر من catch في البرنامج .
٣- Finally / هي اختيارية تستخدم لتنفيذ كود معين ينفذه المفسر إجباريا سوى حدث استثناء أم لم يحدث .
٤- Throw / إعلان حالة الطوارئ هي الوسيلة التي تستخدمها الآلة التخيلية في الجافا للإعلان عن وجود مشكلة أو خطأ في تشغيل البرنامج .
٥- Throws / التصريح على الاستثناء .

الهيكل العام للاستثناءات

```
try{
---------------
جمل مشابهه للخطاء المتوقع
---------------
}catch( نوع الاستثناء    اسم له ){المعالجة المناسبة
}catch(نوع أخر    اسم له ){المعالجة المناسبة
}finally{ شيء يراد تنفيذه إجباريا }
```

لنحاول إضافة الاستثناء في كود القسمة على صفر

```
class aldopaee {
 public static void main(String[] args) {
 int i=0;
 try{
    i=i/i;
System.out.println("star");
}catch(ArithmeticException e){System.out.println("div by zero"+" "+e);
}finally{System.out.println("End");}
 }
```

```
D:\JCreator LE\GE2001.exe
div by zerojava.lang.ArithmeticException: / by zero
End
```

نلاحظ أنة بعد حصول الخطاء تم تجاهل أمر الطباعة وذهب مباشرة ليبحث عن catch المناسب ليعالج هذا الخطاء ومن خلال الشكل يتضح البرنامج .

- إذا حصل الاستثناء المعالجة المناسبة في catch الأول أو الثاني فأنة ينفذ محتواة ثم يذهب إلى finally مباشرة ثم يخرج من try ولا ينفذ catch أخر إطلاقاً .

معالجة الاستثناءات عن طريق الكلمة المحجوزة throws

```
1  class aldopaee {
2  static void compute(int a[]) throws ArrayIndexOutOfBoundsException{
3  System.out.println(a[2]);
4  System.out.println("Normal exit");
5  }
6  public static void main(String args[]) {
7        int a[]={1,2};
8  try {
9  compute(a);
10 } catch (ArithmeticException e) {
11             System.out.println("div by zero " + e);
12 }catch (ArrayIndexOutOfBoundsException e) {
13             System.out.println("Caught " + e);}
14 }
15 }
```

```
D:\JCreator LE\GE2001.exe
Caught java.lang.ArrayIndexOutOfBoundsException: 2
```

في السطر ٢ تم التصريح برأس الدالة عن الخطاء المتوقع .
نلاحظ في السطر السابع تم تعريف مصفوفة من موقعين فقط وفي السطر التاسع يتم الانتقال إلى كود الدالة ويتم في السطر الثالث طباعة محتوى المصفوفة ذات الموقع الثالث وبما أن المصفوفة مكونة ن موقعين إذن فأنة يحصل استثناء فينتقل التنفيذ إلى السطر العاشر ويسأل هل نوع catch من نوع الخطاء فأنة هنا لا يوافق الخطاء فينتقل إلى السطر ١٢ ويحصل تطابق فينفذ محتوى catch ويخرج من try .

- وهذا كود أخر

```
class aldopaee {
  public static void main (String arg[]){
    int denom[] = {2, 0, 0, 4};
try {
     for (int i =0; i< 5; i++) {
       try {
        System.out.println( i+"/"+ denom[i]+"is "+i/denom[i]);
          } catch (ArithmeticException e){
            System.out.println("Can't divide by ZERO!");
                            }
            finally{System.out.println(i+"finally by 1");}
                  }
          } catch (ArrayIndexOutOfBoundsException ex) {
              System.out.println("No matching element found.");
                                          }
            finally{System.out.println("finally by 2\n");}
}
}
```

```
D:\JCreator LE\GE2001.exe
0/2is 0
0finally by 1
Can't divide by ZERO!
1finally by 1
Can't divide by ZERO!
2finally by 1
3/4is 0
3finally by 1
4finally by 1
No matching element found.
finally by 2
```

نلاحظ هنا انه يوجد try بداخل try وسينفذ try الداخلي بعدد مرات اللوب وان تحقق طبع الخطاء الموضح ثم يطبع finally by 1 بعدد مرات اللوب ثم ينتهي بحصول خطاء المصفوفة فيطبع رسالة الخطاء للمصفوفة ثم يطبع finally الخارجي .

- إطلاق الاستثناء .

```
1 class aldopaee {
2 public static void main (String arg[]){
3 try {System.out.println("main+");
4 int i = 0;
5 if (i <= 0)throw new Exception("Throw an error");
6 System.out.println("main-");
7 } catch (Exception e) {System.out.println(e);
8 } finally {System.out.println("Finally block!");}
9 System.out.println("Finished!");}
10 }
```

D:\JCreator LE\GE2001.exe

```
main+
java.lang.Exception: Throw an error
Finally block!
Finished!
```

نلاحظ في السطر 5 انه تم تنشيط أو إطلاق استثناء من نوع أب أي عام وثم الانتقال إلى السطر 7 لمعالجة هذا الاستثناء ثم تنفيذ محتوى finally و تنفيذ بقية الكود .

- إعادة إطلاق الاستثناء

```
1  class aldopaee {
2  public static void method1() throws Exception{
3  try {
4  System.out.println("method1+");
5  throw new Exception();
6  } catch (Exception e) {System.out.println("rethrow an exception");
7  throw e;
8  } finally {System.out.println("finally block1");}
9  }
10 public static void main (String arg[]){
11 try {
12 System.out.println("main+");
13 method1();
14 System.out.println("main-");
15 } catch (Exception e) {System.out.println("caught from main");}
16 finally {System.out.println("finally block2");}
17 System.out.println("fnished");
18 }
19 }
```

```
D:\JCreator LE\GE2001.exe

main+
method1+
rethrow an exception
finally block1
caught from main
finally block2
fnished
```

يبدأ تنفيذ البرنامج من السطر 11 – 13 ثم ينتقل إلى الدالة فينفذ محتواها وفي السطر 5 يتم إطلاق استثناء وتتم معالجته في السطر 6 وفي السطر 7 يعاد إطلاق استثناء من نفس النوع السابق فينفذ محتوى finally الداخلي ثم ينتقل للسطر 15 ليعالج هذا الاستثناء و فينفذ محتوى finally الخارجي ثم بقية كود البرنامج

- قاعدة / عند إطلاق استثناء تم تأتي return فأنها تبطل عمل throw وتخرج من الطريق وهذا الكود شبيه السابق ولا كن الفرق كلمة return .

```
1  class aldopaee {
2  public static void method1() throws Exception{
3  try {
4  System.out.println("method1+");
5  throw new Exception();
6  } catch (Exception e) {System.out.println("rethrow an exception");
7  throw e;
8  } finally {System.out.println("finally block1");return;}
9  }
10
11 public static void main (String arg[]){
12 try {
13 System.out.println("main+");
14 method1();
15 System.out.println("main-");
16 } catch (Exception e) {System.out.println("caught from main");}
17 finally {System.out.println("finally block2");}
18 System.out.println("fnished");
```

```
D:\JCreator LE\GE2001.exe

main+
method1+
rethrow an exception
finally block1
main-
```

نلاحظ انه بعد تنفيذ السطر 8 وجد أمر return

فتم تجاهل الاستثناء الذي أطلق في السطر 7 فانتقل إلى الدالة الرئيسية وتابع تنفيذ بقية الكود بشكل طبيعي .

- إذا استخدمت الكلمة **return** في الدالة الرئيسية فهذا يعني انها البرنامج كلياً بعد ثنفيذ محتوى **finally** وهذا الكود يبين ذلك .

```
1  class aldopaee {
2  public static void main (String arg[]){
3  try {
4  int r2 = 10/0;
5  } catch (ArithmeticException e) {
6  System.out.println("Calculation Error");
7  return;
8  } catch (Exception e) {
9  System.out.println("General Exception");
10 } finally {System.out.println("Finally block");}
11 System.out.println("Finished");
12 }
13 }
```

```
D:\JCreator LE\GE2001.e
Calculation Error
Finally block
```

نلاحظ في السطر 7 وجد المترجم **return** فانتقل الى السطر 10 لينفذ محتوى **finally** وانهى البرنامج .

- من الأخطاء البرمجية الشائعة تقديم **catch** يحتوي على نوع من أنواع الاستثناء أب **Exception**على استثناء ابن **NumberFormatException** كهذا الكود لن ينفذ إطلاقاً .

```
class aldopaee {
public static void main (String arg[]){
try {
    System.out.println("main+");
   int i = 0;
   i=i/1;
} catch (Exception e) {System.out.println(e);
} catch (NumberFormatException e) {System.out.println(e);}
}
}
```

قواعد مهمة

- إذا وجد المترجم في try الخارجي خطاء والمعالجة المناسبة في try الداخلي فأنة لا يدخل إلى try الداخلي إطلاقا ولا حتى finally تبعة بل يذهب مباشرة إلى catch الخاص بة و ينفذ المعالجة المناسبة .

```
class aldopaee {
public static void main (String arg[]){
try {
    int r2 = 10/0;
    try{
     //------
     } catch (ArithmeticException e){
              System.out.println("ArithmeticException");
     }finally {System.out.println("Finally block local");}

   } catch (NullPointerException e)
   {System.out.println("NullPointerException");
} finally {System.out.println("Finally block");}
System.out.println("Finished");
}
}
```

```
D:\JCreator LE\GE2001.exe
Finally block
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at aldopaee.main(aldopaee.java:4)
```

- إن لم يجد المعالجة المناسبة فان لغة الجافا تتكفل بالمسئولية وتطبع نوع الخطاء والاستثناء المناسب لها وكودنا السابق يبين ذلك .
- إذا وجد المترجم في try الداخلي خطاء ولم يلقي المعالجة المناسبة فأنة ينفذ فقط finally الخاص به ثم يخرج إلى try الخارجي ليبحث المعالجة الناسبة وينفذ finally ل try الخارجي .

```
class aldopaee {
public static void main (String arg[]){
try {

    try{
     int r2 = 10/0;
     } catch (NullPointerException e)
           {System.out.println("NullPointerException");
     } finally {System.out.println("Finally block local");}

   } catch (ArithmeticException e){
              System.out.println("ArithmeticException");
   }finally {System.out.println("Finally block ");}
System.out.println("Finished");
}
}
```



- خلاصة القول try الداخلي يخرج إلى الخارجي وليس العكس .

• يراد منك استثناء يعرف هل للعدد جذر صحيح أم لا فان كان للعدد المدخل جذر عشري فأنة يطلق استثناء وان لا تأتي بجذر العدد المدخل ؟

```
import java.io.*;
class aldopaee{
static void check(int a)throws Exception{
   int c=0;
   for (int i = 1; i <= a / 2; i++)
   if (i * i == a)c=i;
 if(c==0)throw new Exception();
 System.out.println("SQRT = "+c);
}
public static void main(String k[])throws IOException{
int a;
BufferedReader b = new BufferedReader(new InputStreamReader(System.in));
      a=Integer.parseInt(b.readLine());
 try{
   check(a);
   }catch(Exception e){System.out.println("NOT SQRT NUMBER");}
   finally{System.out.println("BY");}
}
}
```

```
D:\JCreator LE\GE2001.exe
3
NOT SQRT NUMBER
BY
Press any key to cont
D:\JCreator LE\GE2001.exe
5
NOT SQRT NUMBER
BY
```

• عندما المترجم يوجه كلمه System.exit(); فانة يخرج من البرنامج ولا ينفذ شيء حتى finally وهذا الكود يبين ذلك .

```
class aldopaee{
public static void main(String k[]){
       int i=3;
System.out.println("star");
 try{
   if(i%2!=0){System.exit(1);}
   }catch(Exception e){System.out.println("aldopaee");
   }finally{System.out.println("BY");}
System.out.println("finsh");
}
}
```

```
D:\JCr
star
```

إنشاء استثناء خاص بك

```
class MyException extends Exception {
private int detail;
MyException(int a) {detail = a;}
public String toString() {return "MyException[" + detail + "]";}
}
class aldopaee {
static void compute(int a) throws MyException {
System.out.println("Called compute(" + a + ")");
if(a > 10) throw new MyException(a);
System.out.println("Normal exit");
}
public static void main(String args[]) {
try {
compute(1);
compute(20);
} catch (MyException e) {
System.out.println("Caught " + e);
}
}
}
```

عمل توريث للكلاس من كلاس الاستثناء .

دالة خاصة بمكتبة الاستثناء تنفذ تلقائياً حال حصول تنشيط للاستثناء

```
D:\JCreator LE\GE2001.exe
Called compute(1)
Normal exit
Called compute(20)
Caught MyException[20]
```

- قاعدة / عند حصول خطاء داخل catch فإن المترجم لا يطلع ولا ينزل للبحث عن المعالجة المناسبة بل يطبع نوع الخطاء من داخل لغة الجافا وينفذ محتوى الفاينل وينهي البرنامج وهذا الكود يبين ذلك .

```
class aldopaee{
public static void main(String k[]){
    int i=0,b[]={2,3};
System.out.println("star");
 try{
    i/=i;
  }catch(ArrayIndexOutOfBoundsException e){System.out.println("aldopaee1");
  }catch(ArithmeticException e){System.out.println("aldopaee2");b[2]=i;
  }catch(Exception e){System.out.println("aldopaee3");
  }finally{System.out.println("BY");}
System.out.println("finsh");
}
}
```

```
D:\JCreator LE\GE2001.exe
star
aldopaee2
BY
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 2
        at aldopaee.main(aldopaee.java:10)
```

- قاعدة / إذا أنشأت استثناء خاص بك واحتوى على داله بناء فان إذا حصل إطلاق لهذا الاستثناء داخل البرنامج فان محتوى دالة البناء للاستثناء تنفذ أولاً ثم ما بداخل catch ثانياً

```
class TestException extends Exception {
TestException(){System.out.println("TestExce");}
  }
class aldopaee {
      public static void main(String[] args) {
                 String arg="t";
                 System.out.println(arg);
        try {
           thrower(arg);
            System.out.println("Test ");
           }catch (Exception e) {System.out.println("Test 1");}
         }
     static void thrower(String s) throws TestException {
     try {
        if (s.equals("d")) {int i = 0; i/=i;}
        if (s.equals("t"))throw new TestException();
         }catch (TestException e) {System.out.println("Test 2");}
         }
}
```

```
D:\JCreator L
d
Test 1
Press any ke

D:\JCreator
t
TestExce
Test 2
Test
```

- إذا حصل إطلاق للاستثناء الخاص بك فأنة يذهب إلى الاستثناء وينفذ محتواة ثم ينفذ محتوى catch .
- نلاحظ عند عدم إطلاق الاستثناء الخاص بك فأنة يبحث عن catch المناسب في البرنامج الرئيسي ويعالجه ولا علاقة له بالكلاس الخاص بك وما بداخلة من دوال بناء وغيرة .

- سؤال للطالب النبيل فقط

من المعروف انه عند حصول إطلاق للاستثناء داخل ألداله فان المترجم يوقف ايعازات محتوى الدالة ويذهب مباشرة للبحث عن catch المناسب خارج الدالة مثل هذا الكود .
السؤال هنا أنا أريد تنفيذ محتوى ألداله كاملاً سوى حصل إطلاق للاستثناء أم لا ففي هذا الكود سيتم تجاهل تنفيذ أمر الطباعة الموجود في السطر B فما الحل ؟
بشرط بدون استخدام finally
فيكون الناتج هكذا

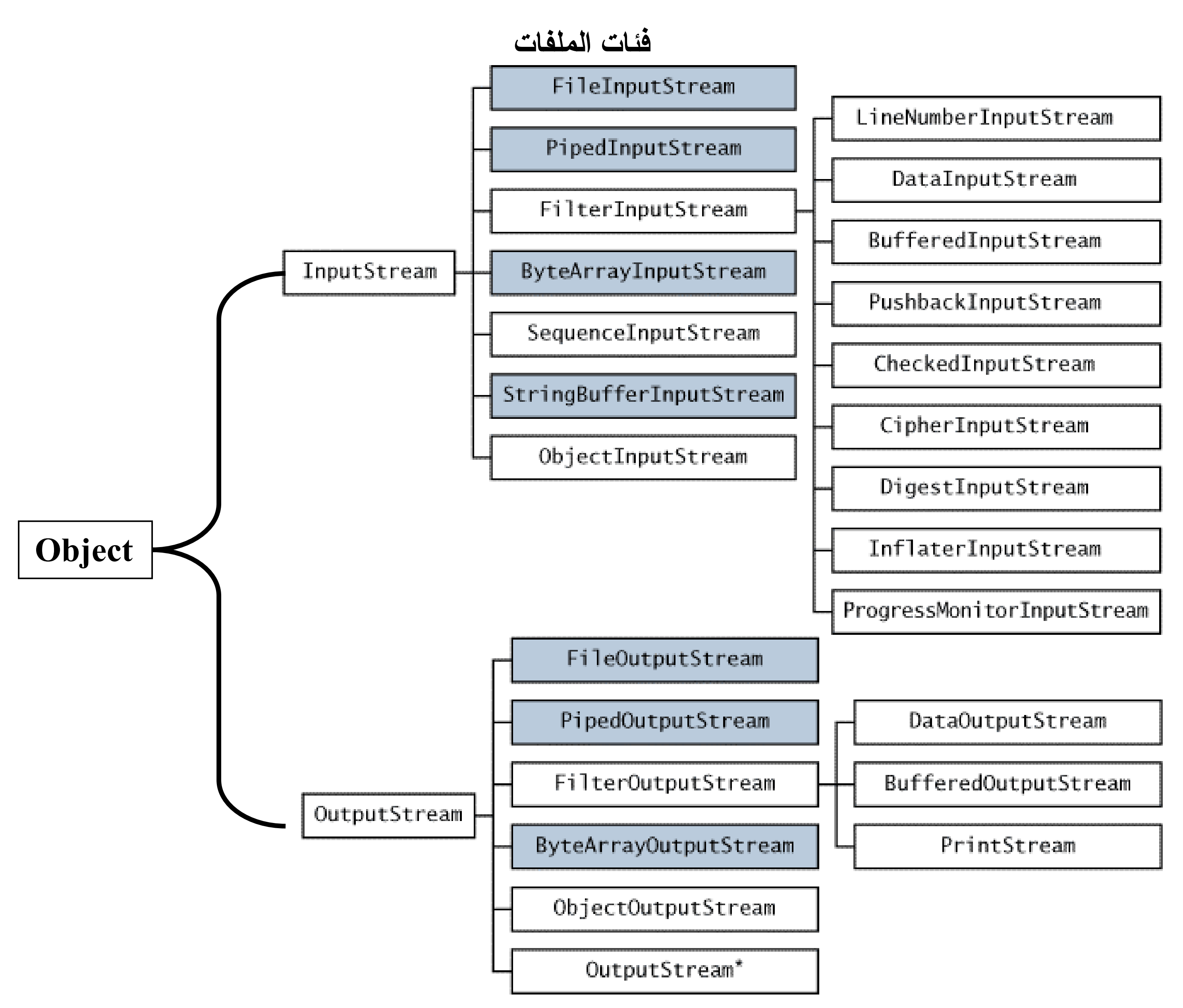
```
class aldopaee {
     public static void main(String[] args) {
             String arg="ammar";
         try {
               thrower(arg);
            }catch (Exception e) {
                System.out.println("ammar 1");}
                          }
static void thrower(String s) throws Exception {
 try {
     if (s.equals("ammar")){int i = 0; i/=i;}
      }catch (Exception e){
              System.out.println("ammar 2");throw  e;}

  B: System.out.println("ammar aldopaee");
                                            }
}
```

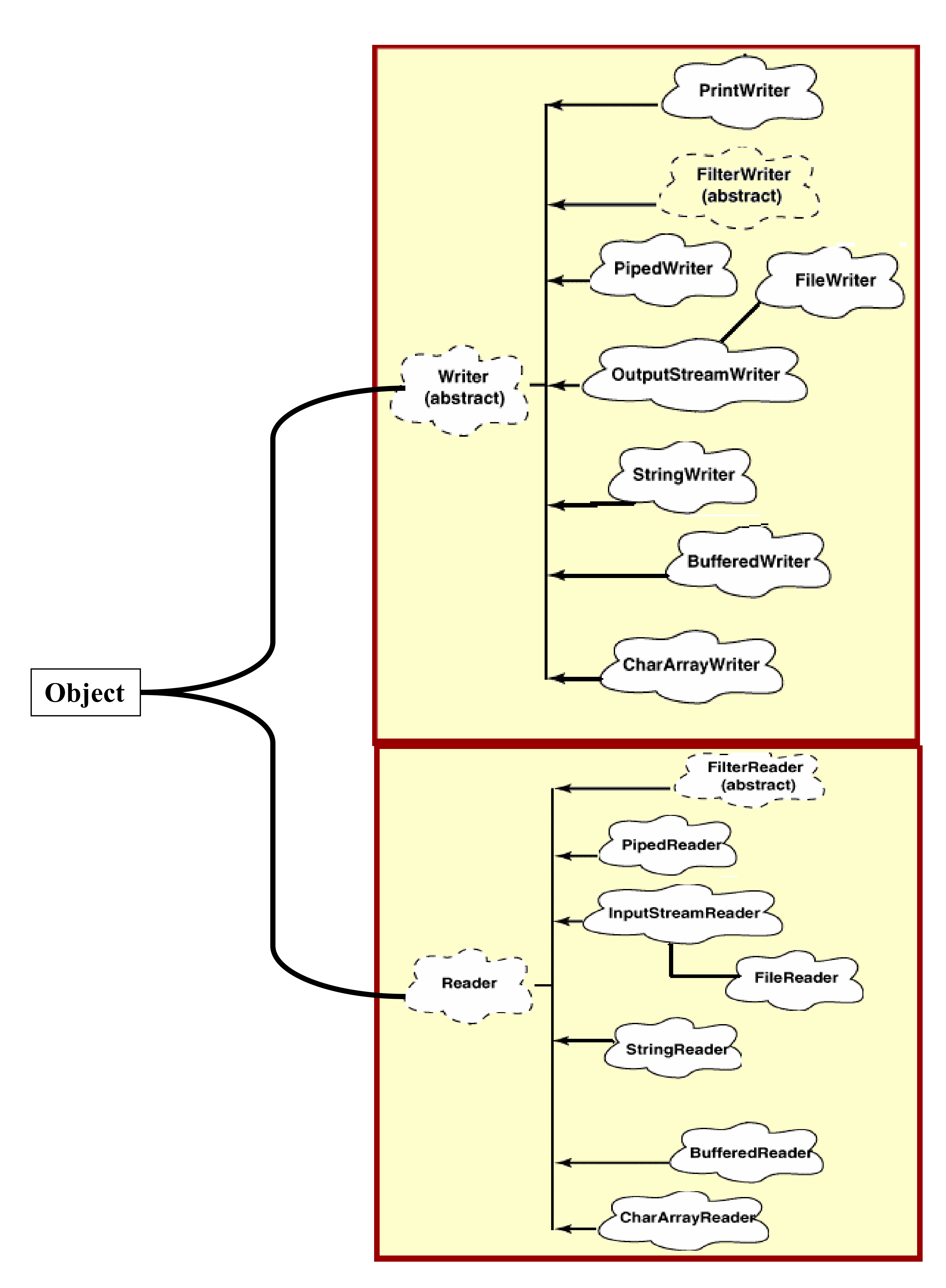


الملفات File

كثيراً ما يستصعب مبتدئو البرمجة مواضع التعامل مع الملفات ، والأمر ليس لصعوبة الموضوع لحد ذاته بل إلى عرضة والطريقة التي يحاول فيها المبتدئ التعامل مع الموضوع فهو ربما أنهى أصعب مواضيع البرمجة مبدئياً ، وربما في احد الأيام أراد تطوير برنامجه ليكون قادراً على التعامل مع الملفات وحتى يفعل ذلك فأنه لا يأخذ هذا الموضوع بشكل جدي ويتجاوز أساسياته ليذهب بعيداً كي يتعامل مع المواضيع المتقدمة نسبياً والنتيجة لا شيء عدا إضاعة الوقت فيما لا يجدي وحتى تكون قادراً على فهم هذه الوحدة فأرجو منك أن تتعامل معها على أنها وحدة متكاملة لها أساسياتها الأولية وما إلى ذلك ولا تتعامل معها على أنها وحدة أمثلة تطبيقية فحسب .



تتعامل الفئات **inputstream , outputstream** والفئات المشتقة منها مع مجاري دخل البايت .



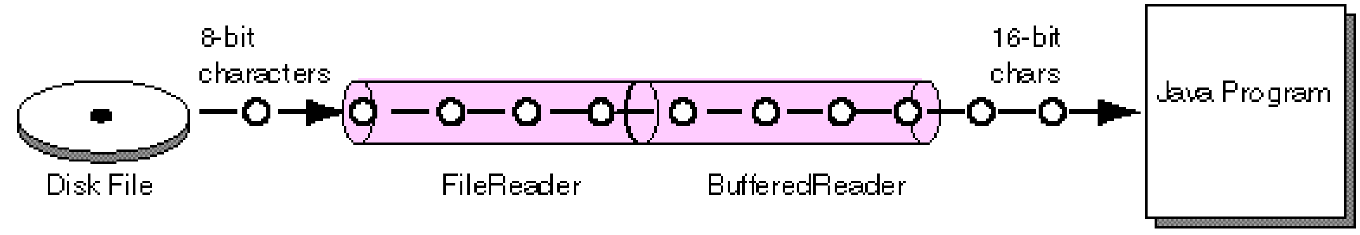
تتعامل الفئات Reader , Write والفئات المشتقة منها مع مجاري دخل من المحارف .

عمليات الملفات

من العمليات التي تجرى على الملفات عملية القراءة والكتابة فأفضل طرق اتبعتها وأحببتها وهي

- **القراءة من ملف**

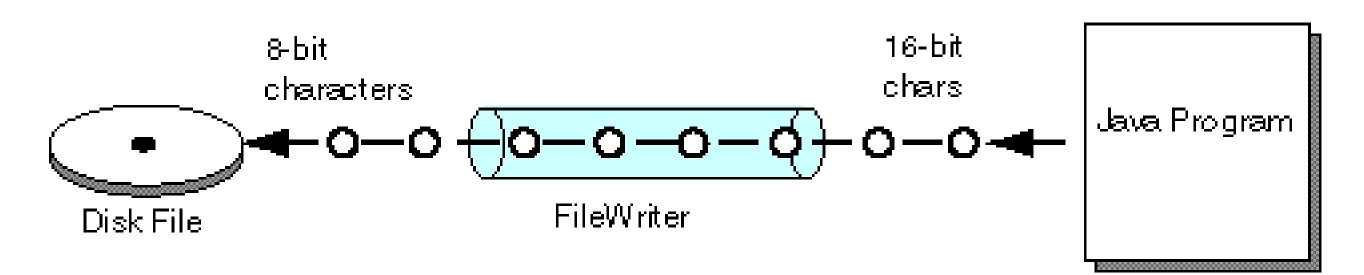
```
1-  BufferedReader br = new BufferedReader(new
    FileReader("c:\\ammar.txt"));
```




```
2-  FileReader br=new FileReader("c:\\ammar.txt"));
3-  FileInputStream br=new
    FileInputStream("c:\\ammar.txt"));
```

- **الكتابة على الملف**

```
1-  PrintWriter br=new PrintWriter(new
        FileWrite("c:\\ammar.txt"));
     Br.println("ammar");
2-  FileOutputStream br=new
        FileOutputStream("c:\\ammar.txt"));
     Br.write("ammar");
3-  FileWriter br= new FileWriter("c:\\ammar.txt"));
     Br.write("ammar");
```



وكما بينا سابقا أن عملية القراءة تتمثل في كيفية كتابة تعريف القراءة كالقراءة على بايت أو على سلسلة محارف .

```
Int c=(char)br.read();        ——►  على حرف حرف
String c=br.readLine();       ——►  على سطر سطر
```

- وهذا الكود يبين كيفية خزن عشره أعداد عشوائية في ملف

```
class aldopaee {
public static void main(String[] args){
java.util.Random a=new java.util.Random();
try{
   PrintWriter out=new PrintWriter(new FileWriter("c:\\ aldopaee.txt",false));
   for (int i=0;i<10;i++)
   out.println(a.nextInt(100));
   out.close();
   }catch(IOException e){System.out.println("File Not Found ");}
System.out.println("completed insert number");
}
}
```

لو عملت **true** فإننا سنضيف فوق الملف أى إضافة .

ألان بعد تنفيذ الكود اذهب إلى القرص السي وستجد ملف باسم **aldopaee** افتح الملف وستجد العشرة الأعداد التي ولدتها بواسطة الدالة **random** موجودة داخل الملف .

- عند عملية قراءة من ملف أو إضافة فيجب استخدام الاستثناءات .
- هذا الكود يقرءاه محتويات الملف السابق بصيغه محارف

```
import java.io.*;
public class aldopaee {
   public static void main(String[] args)throws IOException {
    BufferedReader fi;
    String s;
try {
 fi = new BufferedReader(new FileReader ("c:\\aldopaee.txt"));

while ((s=fi.readLine()) != null)
System.out.println(s);
}catch (Exception e) { System.err.println("File Not Found ");}

  }
}
```

- نلاحظ انه تم قراءه محتويات الملف بصيغة سلسلة وان أردت إجراء بعض العمليات الحسابية كمجموع أو اكبر قيمة أو ما شابة ذلك فيجب تحويل من صيغة سلسلة رقمية إلى أرقام بواسطة الدالة **Integer.parsInt()** وقد تم شرحها سابقاً .

- وهذا كود لقراءة الملف السابق على حرف حرف ومعرفة عدد السطور

```
import java.io.*;
class aldopaee {
public static void main(String args[])throws IOException
```

```
{
int i,g=0;String s;
FileInputStream fin;
try {
    fin = new FileInputStream("c:\\aldopaee.txt");
   } catch(FileNotFoundException e){System.out.println("File Not Found");
return;
}

do{
i = fin.read();
if(i!=-1)System.out.print((char) i);
if(i==10)g++;
}while(i != -1);
System.out.println("The Number Of line  : "+g);
fin.close();
}
}
```

إذا كان I= شفرة enter فإننا انتقلنا إلى سطر جديد

طالما لم نصل إلى نهاية الملف

- في كودنا هذا بإمكاننا قراءة الملف على سطر سطر باستبدال

```
while((s = fin.readLine())!=null)
```

- وهذا كود يعمل على عد كلمات ملف وعدد الأسطر

```
import java.io.*;
public class aldopaee {
   public static void main(String[] args)throws IOException {
     BufferedReader fi;
     int word = 0,line=0;String s;int i;
try {
 fi = new BufferedReader(new FileReader ("c:\\aldopaee.txt"));

while ((s=fi.readLine()) != null)
{s=s+" |";line++;
     for(i=0;i<s.length()-1;i++){
          if(s.charAt(i)==' '&s.charAt(i+1)!=' ')word++;
               System.out.print(s.charAt(i)); }
System.out.println();
}
System.out.println("line = "+line+"    word= "+word);
}catch (Exception e) { System.err.println("File Not Found ");}

  }
}
```

- وهذا الكود يفتح ملف وينسخه إلى ملف أخر بدون مسح محتويات الملف الثاني أي إضافة

```
import java.io.*;
class aldopaee {
public static void main(String args[])throws IOException
```

```
{
int g=0;
FileInputStream fin;
FileOutputStream fin2=new FileOutputStream("c:\\ aldopaee.txt",true);
try {
     fin = new FileInputStream("c:\\ out.txt");
     } catch(FileNotFoundException e) {System.out.println("File Not Found");return;}

while(g != -1){
g = fin.read();
if(g!=-1)fin2.write((char)g);
}
fin.close();fin2.close();
System.out.println("Copy File Good");
}
}
```

المصادر

- الطريق إلى احتراف الجافا / عزب محمد عزب .
- Osborne - Java 2--Complete Reference (5th Ed 2002)
- Thinking in Java, 2nd edition, Revision 12
- المؤسسة العامة لتعليم الفني والتدريب المهني – جافا متقدم
- بعض مواقع الويب

١- الموسوعة العربية http://www.c4arab.com

٢- javagirl

٣- المدرس العربي

٤- منتديات الفريق العربي للبرمجة


الخاتمة
إلى كل من انتفع بهذا الكتاب
إلى كل مسلم ومسلمة
إلى كل من يشهد أن لإله إلا الله محمد رسول الله
ارجوا الدعاء لي ولمن يصلي على النبي الأمي
صلوات الله وسلامه عليك يا حبيبي محمد ابن
عبد لله
والحمد لله.

من مؤلفاتنا


DATA STRUCTURES
C#
هياكل البيانات


أصول البرمجة
بلغة C++
aldopaee@hotmail.com


مئة سؤال و سؤال
لإتقان البرمجة
aldopaee@hotmail.com
2006 - عام الإصدار


OOP IN JAVA
البرمجة الموجهة بلغة
OOP IN JAVA
JAVA